51CTO.com
技术成就梦想
我 们 只 谈 开 发

开发月刊

Development Monthly

2013年05月
总第026期

R语言学习笔记(1)：R是什么

数据迁移：高难度带来高回报

创业像水墨画：三千世界 致在桥上看风景的你

编程排行 Billboard

专题报道 Special report

技术热点 Techlogy hot

编者按

2013年5月9日，Tiobe公司发布新一期编程语言排行榜。新一期榜单前10位没有太多的变化，只是Objective-C与C++，Ruby与JavsScript在互相交换位置罢了。今天我们要关注的是排在TOP 20后半部的一门语言——Bash。

2013年5月编程语言排行榜：UNIX下的Bash

2013年5月9日，Tiobe公司发布新一期编程语言排行榜。新一期榜单前10位没有太多的变化，只是Objective-C与C++，Ruby与JavsScript在互相交换位置罢了。今天我们要关注的是排在TOP 20后半部的一门语言——Bash。

大家先请看本期TOP20榜单

Position May 2013	Position May 2012	Delta in Position	Programming Language	Ratings May 2013	Delta May 2012	Status
1	1	=	C	18.729%	+1.38%	A
2	2	=	Java	16.914%	+0.31%	A
3	4	↑	Objective-C	10.428%	+2.12%	A
4	3	↓	C++	9.198%	-0.63%	A
5	5	=	C#	6.119%	-0.70%	A
6	6	=	PHP	5.784%	+0.07%	A
7	7	=	(Visual) Basic	4.656%	-0.80%	A
8	8	=	Python	4.322%	+0.50%	A
9	9	=	Perl	2.276%	-0.53%	A
10	11	↑	Ruby	1.670%	+0.22%	A
11	10	↓	JavaScript	1.536%	-0.60%	A
12	12	=	Visual Basic .NET	1.131%	-0.14%	A
13	15	↑↑	Lisp	0.894%	-0.05%	A
14	18	↑↑↑↑	Transact-SQL	0.819%	+0.16%	A
15	17	↑↑	Pascal	0.805%	0.00%	A
16	24	↑↑↑↑↑↑↑↑	Bash	0.792%	+0.33%	A
17	14	↓↓↓	Delphi/Object Pascal	0.731%	-0.27%	A
18	13	↓↓↓↓↓	PL/SQL	0.708%	-0.41%	A
19	22	↑↑↑	Assembly	0.638%	+0.12%	B
20	20	=	Lua	0.632%	+0.07%	B

从2013年4月的编程语言排行榜我们惊异的发现Bash这门UNIX下的壳语言，竟然有了飞速的上升。从第34位最高上升到第13位。究竟这门语言有什么独特之处?

Bash的诞生

Bash这个单词的来源十分晦涩，它的名字是一系列缩写：Bourne-Again SHell — 这是关于Bourne shell（sh）一个双关语（Bourne again / born again）。

Bash语言之父Stephen R. Bourne

Bash是大多数Linux系统以及Mac OS X v10.4默认的shell，它能运行于大多数Unix风格的操作系统之上，甚至被移植到了Microsoft Windows上的Cygwin系统中，以实现windows的POSIX虚拟接口。此外，它也被DJGPP项目移植到了MS-DOS上。

Bash是Linux的敲门砖，如果不懂Bash那其他东西就不用学习了。Linux透过终端下达指令，都是通过Bash来执行。在旧金山的独立开发者Sara Mei就一直用Ruby开发大量的应用，在他看来，Ruby的魅力在于它是一个使用起来非常舒服的编程语言，它具有许多强大的元编程功能。

Bash语法

用vi编辑器编辑一个hello文件如下：

```
#!/bin/bash
# This is a very simple example
echo Hello World
```

这样最简单的一个 BASH 程序就编写完了。这里有几个问题需要说明一下：

一，第一行的 #! 是什么意思

二，第一行的 /bin/bash 又是什么意思

三，第二行是注释吗

四，echo 语句

如何执行该程序呢？有两种方法：一种是显式制定 BASH 去执行：

$ bash hello 或$ sh hello（这里 sh 是指向 bash 的一个链接，“lrwxrwxrwx 1 root root 4 Aug 20 05:41 / bin/sh -> bash”）

或者可以先将 hello 文件改为可以执行的文件，然后直接运行它，此时由于 hello 文件第一行的“#! /bin/bash” 的作用，系统会自动用/bin/bash 程序去解释执行 hello 文件的：

```
$ chmod u+x hello
$ ./hello
```

此处没有直接 “$ hello”是因为当前目录不是当前用户可执行文件的默认目录，而将当前目录“.”设为默认目录是一个不安全的设置。

需要注意的是，BASH 程序被执行后，实际上 Linux 系统是另外开设了一个进程来运行的。

Bash命令行参数

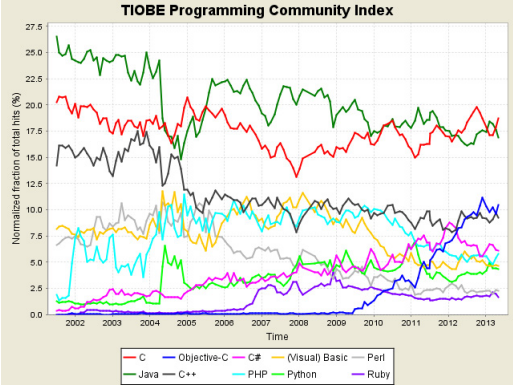
在使用wget的时候，我喜欢加上一个-c的参数，这样可以让wget启用断点续传功能。这里的-c就是一个命令行参数。

在写c语言的时候，我们使用字符串数组存储命令行参数，像我们所熟悉的argv[]。当然我们也需要命令行参数的个数，这存储在名为argc的整型变量中。argc和argv是约定俗成的名称，当然你可以使用自己的名称作为main函数的形参。■

本文未完，更多部分请参考原文：

http://developer.51cto.com/art/201305/392833.htm

附：编程语言长期趋势



50到100位编程语言

(Visual) FoxPro, ABC, Agilent VEE, Algol, Alice, Apex, ATLAS, AutoLISP, bc, BlitzMax, C shell, CFML, CL (OS/400), Clarion, Clipper, Clojure, Dart, Dylan, Eiffel, Emacs Lisp, Fantom, Gambas, Go, Groovy, Heron, Icon, IDL, Informix-4GL, J, JavaFX Script, Lasso, LPC, MUMPS, Oberon, OCaml, Occam, OpenCL, Oz, Pike, PowerShell, REXX, S, sed, SPARK, thinBasic, VBScript, VHDL, WebDNA, xBase, XSLT

编程语言类别

Category	Ratings May 2013	Delta May 2012
Object-Oriented Languages	58.2%	+0.8%
Procedural Languages	37.0%	+0.1%
Functional Languages	3.2%	-0.7%
Logical Languages	1.7%	-0.2%

Category	Ratings May 2013	Delta May 2012
Statically Typed Languages	70.6%	-0.7%
Dynamically Typed Languages	29.4%	+0.7%

出版方：

北京无忧创想信息技术有限公司

责任编辑：

彭凡　林师授

封面设计：

苍旭

联系方法：

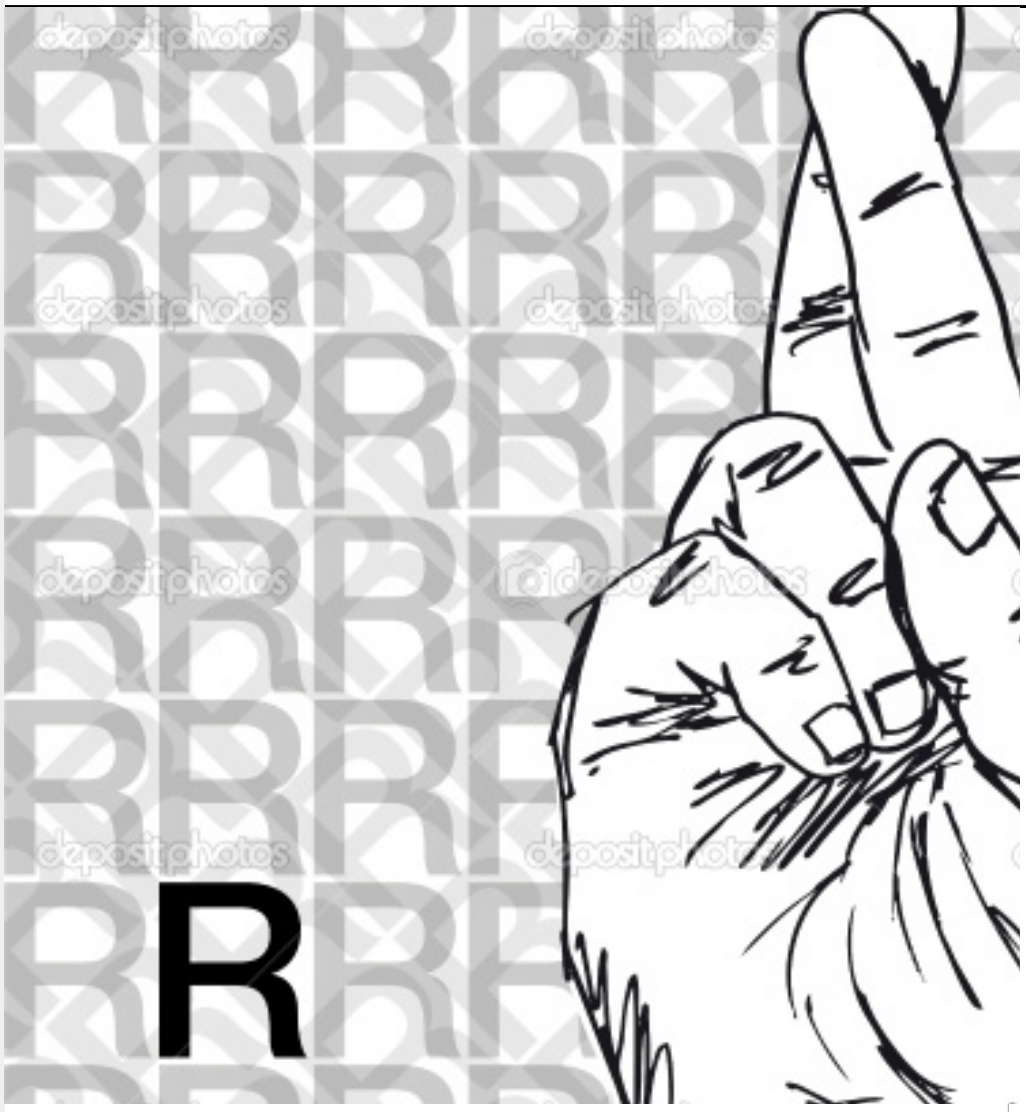
邮箱：linss@51cto.com

电话：010-68476606-8123

出版日期：2013年05月

欢迎来稿，请发送邮件至

linss@51cto.com



今天我们将为大家带来的是有关R语言的技术分享。

R是用于统计分析、绘图的语言和操作环境。R是属于GNU系统的一个自由、免费、源代码开放的软件，它是一个用于统计计算和统计制图的优秀工具。

透过本期《开发月刊》R语言主题，您不光对R语言的有一定的了解，对其实际工作案例，也会有更深的认识。

51CTO开发频道将会继续关注R语言的发展，为大家带来更多有关R语言的分享。

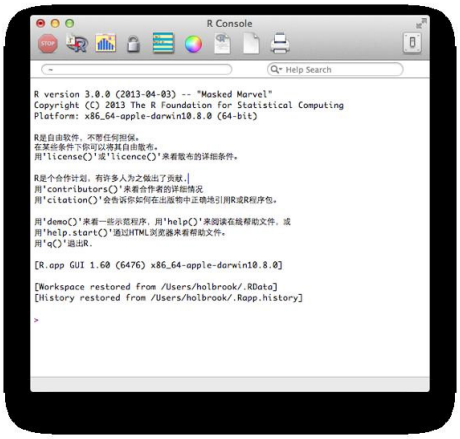
51CTO开发频道寄语

R语言学习笔记(1)：R是什么

1. R初窥

从CRAN（The Comprehensive R Archive Network）cran.r-project.org—mirrors.html中选择一个镜像，然后下载合适的安装包（R支持Linux、Mac OS X和Windows）。

安装并运行R后，可以看到R的控制台（我的操作系统是Mac OS）：



在R的控制台输入如下命令：

> install.packages(‘quantmod’) # 安装quantmod包

> require(quantmod) #引用quantmod包

> getSymbols(“GOOG”,src=”yahoo”,from=”2013-01-01”, to=’2013-04-24’) #从雅虎财经获取google的股票数据

> chartSeries(GOOG,up.col=’red’,dn.col=’green’) #显示K线图 > addMACD() #增加MACD图

就能够看到下图的效果了：

最后，退出R：

```
> q()#Terminate an R Session
```

2. R是什么

是不是很神奇？反正当时我完全被Hold住了。

那么R到底是什么？或者说，R到底是做什么用的？从不同的角度出发，对R会有不同的描述。

从使用角度，R是一个有着统计分析功能及强大作图功能的软件，在GNU协议General Public Licence4下免费发行。

从编程角度，R语言是面向对象的统计编程语言，是由AT&T贝尔实验室所创的S语言发展出的一种方言。

从计算角度，R是一种为统计计算和图形显示而设计的语言及环境。

从开发角度，R是一组开源的数据操作，计算和图形显示工具整合包有各种方式可进行编程调用。

从架构角度，R是为统计计算和图形展示而设计的一个系统。它包括一种编程语言，高级别图形展示函数，和其它语言的接口以及调试工具。

如果一定要找到一个与R类似的软件，那就是商业软件Matlab。R和Matlab都是基于编程进行数据分析的工具，Matlab适用的领域更广，而R更擅长统计分析领域。

与Matlab相比，R更具备开放性：

R是自由软件，Matlab是商业软件；

R可以方便的通过“包”进行扩展，R的核心只有25个包，但是有几千个外部包可以调用，当然你也可以开发自己的；

R语言比Matlab的要强大；

R和其他编程语言/数据库之间有很好的接口；其他语言也可以很方便的调用R的API和结果对象。

R常用于金融和统计领域。大多数人使用R就是因为它的统计功能，R的内部实现了很多经典的or时髦的统计技术。

3. R的核心概念

3.1 对象

R语言是一种面向对象的语言，所有的对象都有两个内在属性：元素类型和长度。

元素类型是对象内元素的基本类型，包括：数值(numeric),字符型(character),复数型(complex)、逻辑型（logical）、函数（function）等，通过mode()函数可以查看一个对象的类型。

长度是对象中元素的数目，通过函数length()可以查看对象的长度。

除了元素类型外，对象本身也有不同的“类型”，表示不同的数据结构（struct)。R中的对象类型主要包括：

向量(vector): 由一系列有序元素构成。

因子(factor）：对同长的其他向量元素进行分类（分组）的向量对象。R 同时提供有序（ordered）和无序（unordered）因子。

数组(array)：带有多个下标的类型相同的元素的集合

矩阵(matrix)：矩阵仅仅是一个双下标的数组。R提供了一下函数专门处理二维数组（矩阵）。

数据框(data frame)：和矩阵类似的一种结构。在数据框中，列可以是不同的对象。

时间序列(time series)：包含一些额外的属性,如频率和时间.

列表（list）:是一种泛化（general form）的向量。它没有要求所有元素是同一类型，许多时候就是向量和列表类型。列表为统计计算的结果返回提供了一种便利的方法。

3.2 常量

R中还定义了一些常量，比如：

NA：表示不可用

Inf: 无穷

-Inf: 负无穷

TRUE：真

FALSE：假

■

本文未完，更多内容请看原文：
http://developer.51cto.com/art/201305/393121.htm

R语言学习笔记(2)：数据类型和数据结构

■ 尽管前面提到R是面向对象的，但是个人认为R中的所谓对象其实只是一种结构（struct）。还是要使用函数对其进行操作。

R中的数据结构主要面向《线性代数》中的一些概念，如向量、矩阵等。值得注意的是，R中其实没有简单数据（数值型、逻辑型、字符型等），对于简单类型会自动看做长度为1的向量。比如：

```
> b=5
> length(b) [1] 1
> typeof(b) [1] "double"
> mode(b) [1] "numeric"
```

R中最重要的数据结构是向量(vector)和矩阵(matrix)。

向量由一系列类型相同的有序元素构成；矩阵是数组(array)的一个特例：维数为2的数组；而数组又是增加了维度（dim)属性的向量。

除此之外，列表（list）和数据框（data frame）分别是向量和矩阵的泛化——列表允许包含不同类型的元素，甚至可以把对象作为元素；数据框允许每列使用不同类型的元素。对于列表和数据框，其中的元素通常称为分量（components）。

对象的类型和长度

R中所有的对象都有类型和长度属性，可以通过函数typeof()和length()获取/设置。举例如下：

```
> x = c(1,2,3,4) > x [1] 1 2 3 4
> typeof(x) [1] "double"
> length(x) [1] 4
> dim(x)=c(2,2) > x    ,1] [,2] [1,]    1    3
  [2,]    2    4
> typeof(x) [1] "double"
> length(x) [1] 4
> Lst <- list(name="Fred", wife="Mary",
  no.children=3, +                    cchild.
  ages=c(4,7,9)) >
> Lst $name [1] "Fred"  $wife [1] "Mary"
  $no.children [1] 3  $child.ages [1] 4 7 9
> typeof(Lst) [1] "list"
> length(Lst) [1] 4
```

typeof()函数可能返回如下的值（在R源代码src/main/util.c的TypeTable中定义）：

数据对象

logical	含逻辑值的向量
integer	含整数值的向量
double	含实数值的向量
complex	含复数值的向量
character	含字符值的向量
raw	含字节值的向量

其他对象

list	列表

NULL 空

closure 函数

special 不可针对参数求值的内置函数

builtin 可针对参数求值的内置函数

environment 环境

通常在R内部使用

symbol 变量名

pairlist 成对列表对象

promise 用于实现悠闲赋值的对象

language R 语言构建

... 特定变量长度参数

any 可以匹配任何类型的特殊类型

expression 表达式对象

externalptr 外表指针对象

weakref 弱引用对象

char 字符

bytecode 二进制

对象的类型不是一成不变的，可以随时进行转换。接着上面的例子：

```
> typeof(x) [1] "double"
> y = as.logical(x) > typeof(y) [1] "logical"
```

转换的规则如下表：

	to numeric	to logical	to character
from numeric	-	0 → FALSE 其它数字 → TRUE	1, 2, ... → "1", "2"
from logical	FALSE → 0 TRUE → 1	-	TRUE → "TRUE" FALSE → "FALSE"
from character	"1", "2", ... → 1, 2, ... "A",... →NA	"FALSE", "F" → FALSE "TRUE", "T" → TRUE 其它 → NA	-

对象的长度也可以随时发生改变，常见的包括如下情况：

```
> # 扩大索引范围
> x = c(1,2,3) > x [1] 1 2 3
> x[5] = 12
> x [1]  1  2  3 NA 12
> length(x) [1] 5
> # 直接设置length属性
> length(x) = 2
> x [1] 1 2
> # 重新赋值（略）
```

对象的class和attributes typeof()处理对象内元素的类型，而class()处理对象本身的类，例如： > x = 1:6

```
> x [1] 1 2 3 4 5 6
> typeof(x) [1] "integer"
> class(x) [1] "integer"
> dim(x) = c(3,2) > x     [,1] [,2] [1,]   1   4 [2,]   2   5 [3,]   3   6
> typeof(x) [1] "integer"
> class(x) [1] "matrix"
```

■

原文未完，更多内容请参考原文：
http://developer.51cto.com/art/201305/393125.htm

R语言学习笔记(3)绘图

□ 心内求法/文

R提供了非常丰富的绘图功能，可以通过命令：demo (graphics) 或者demo(persp)来体验R绘图功能的强大。

图形工具是 R 环境的一个重要组成部分。 R提供了多种绘图相关的命令，分成三类：

高级绘图命令：在图形设备上产生一个新的图区，它可能包括坐标轴，标签，标题等等。

低级绘图命令：在一个已经存在的图上加上更多的图形元素，如额外的点，线和标签。

交互式图形命令：允许交互式地用鼠标在一个已经存在的图上添加图形信息或者提取图形信息。

在R中执行绘图命令，会启动一个图形设备驱动（device driver）。该驱动会打开特定的图形窗口（graphics window）以显示交互式的图片。一旦设备驱动启动，R 绘图命令可以用来产生统计图或者设计全新的图形显示。此外，R 有一系列图形参数。这些图形参数可修改从而定制你的图形环境。

高级绘图命令

高级图形显示函数可以根据数据显示完整的图形（chart），包括坐标轴，标签、标题、序列等。如果你之前熟悉其他绘图库（比如JFreeChart，matplotlib等）的概念，可以很容易的掌握R中的绘图函数。

图表类型

R支持很多图表类型。在扩展包里面可能会提供更多的图表类型。下表给出R基本环境中支持的图表类型及其对应的绘图函数：

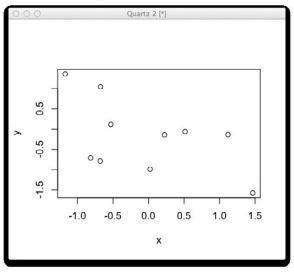
其他包中可能也会提供额外的图表类型，如：

quantmod包提供的K线图：chartSeries（）

portfolio包提供的tree map（板块层级图）：map.market(id, area, group, color)用矩形面积来表示数值，可用于分析磁盘空间占用

lattice包提供的平行坐标图：parallel(data)

```
> x = rnorm(10)
> y = rnorm(10)
> plot(x,y)
```



绘图参数

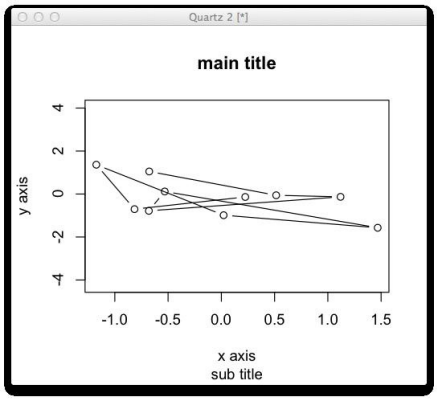
很多时候，你可能需要调整图形的显示方式。R的绘图参数几乎可以定制图形的任何显示（如标题，坐标轴，颜色，字体等）。

R 拥有一个数目很大的图形参数列表。该列表包括控制线条样式，颜色，图形排列和文字对齐等方面的参数。

更改图形参数有两种方式，一种是直接在绘图函数中设置参数，这种方式只影响当前绘图函数，但是不是所有的参数都能够通过这种方式设置；另一种是通过par()函数设置，这种方式会影响当前绘图设备上的所有图形。下面先看一个直接设置参数

的例子：

```
>plot(x,y,type="b",main="main
  title",sub='sub title',xlab="x axis",ylab='y
  axis',asp=0.2)
```



使用par()函数的例子：

```
opar <- par()#备份原来的绘图参数
par(bg="lightyellow", col.axis="blue",
  mar=c(4, 4, 2.5, 0.25))
plot(x, y, xlab="Ten random values",
  ylab="Ten other values",
xlim=c(-2, 2), ylim=c(-2, 2), pch=22,
  col="red", bg="yellow",
bty="l", tcl=-.25, las=1, cex=1.5)
  par(opar)#恢复原来的绘图参数
```

通过?par可以查到所有参数的说明。下面列举最常用的一些参数：

adj控制关于文字的对齐方式,0是左对齐,0.5是居中对齐,1是右对齐,值> 1时对齐位置在文本右边的地方,取负值时对齐位置在文本左边的地方;如果给出两个值(例如c(0, 0)),第二个只控制关于文字基线的垂直调整。

bg指定背景色(例如bg=" red" , bg=" blue" ; 用colors()可以显示657种可用的颜 色名)

bty控制图形边框形状,可用的值为: "o" , "l" , "7" , "c" , "u" 和"]" (边框和字符 的外表相像);如果bty=" n" 则不绘制边框

cex控制缺省状态下符号和文字大小的值;另外,cex.axis控制坐标轴刻度数字大小,cex.lab控制坐标轴标签文字大小,cex.main控制标题文字大小,cex.sub控制副标题文字大小

col控制符号的颜色;和cex类似,还可用:col.axis, col.lab, col.main, col.sub

font控制文字字体的整数(1: 正常,2: 斜体,3: 粗体,4: 粗斜体);和cex类似, 还可用: font.axis, font.lab, font.main, font.sub

las控制坐标轴刻度数字标记方向的整数(0: 平行于轴,1: 横排,2: 垂直于轴,3: 竖排)

lty控制连线的线型,可以是整数(1: 实线,2: 虚线,3: 点线,4: 点虚线,5: 长虚线,6: 双虚线),或者是不超过8个字符的字符串(字符为从" 0" 到" 9" 之间的数字)交替地指定线和空白的长度,单位为磅(points)或象素,例如lty=" 44" 和lty=2效果相同。

lwd控制连线宽度的数字

mar控制图形边空的有4个值的向量c(bottom, left, top, right), 缺省值 为c(5.1, 4.1, 4.1, 2.1)

mfcolc(nr,nc)的向量,分割绘图窗口为nr行nc列的矩阵布局,按列次序使用各子窗口

mfrow同上,但是按行次序使用各子窗口(参照4.1.2)

pch控制符号的类型,可以是1到25的整数,也可以是""里的单个字符

ps控制文字大小的整数,单位为磅(points)

pty指定绘图区域类型的字符,”s”:正方形,”m”:最大利用

tck指定轴上刻度长度的值,单位是百分比,以图形宽、高中最小一个作为基数; 如果tck=1则绘制grid

tcl同上,但以文本行高度为基数(缺省下tcl=−0.5)

xaxt如果xaxt=”n”则设置x−轴但不显示(有助于和axis(side=1, ...)联合使用)

yaxt如果yaxt=”n”则设置y−轴但不显示(有助于和axis(side=2, ...)联合使用)

低级绘图命令

R还可以在现有图形(通过高级绘图命令绘制)的基础上增加一些额外的显示,如标题、绘制坐标轴、在特定的位置增加图形(比如辅助线,拟合线)或文字等。

这些函数在R中称为低级作图命 令(low−level plotting commands)。一些常用低级图形命令包括:

scatter.smooth(x, y, …)LOESS(局部加权散点平滑)拟合曲线

points(x, y)添加点(可以使用选项type=)

lines(x, y)同上,但是添加线

text(x, y, labels,…)在(x,y)处添加用labels指定的文字;典型的用法是: plot(x, y, type=”n”); text(x, y, names)

mtext(text,side=3, line=0,…)在边空添加用text指定的文字,用side指定添加到哪一边(参照 下面的axis());line指定添加的文字距离绘图区域的行数

segments(x0, y0,x1, y1)从(x0,y0)各点到(x1,y1)各点画线段

arrows(x0, y0,x1, y1, angle= 30,code=2)同上但加画箭头,如果code=2则在各(x0,y0)处画箭头,如 果code=1则在各(x1,y1)处画箭头,如果code=3则在两端都画箭头; angle控制箭头轴到箭头边的角度

abline(a,b)绘制斜率为b和截距为a的直线

abline(h=y)在纵坐标y处画水平线

abline(v=x)在横坐标x处画垂直线

abline(lm.obj)画由lm.obj确定的回归线

rect(x1, y1, x2, y2)绘制长方形,(x1, y1)为左下角,(x2,y2)为右上角

polygon(x, y)绘制连接各x,y坐标确定的点的多边形

legend(x, y, legend)在点(x,y)处添加图例,说明内容由legend给定

title()添加标题,也可添加一个副标题

axis(side, vect)画坐标轴,side=1时画在下边,side=2时画在左边,side=3时画在上边,side=4时画在右边。可选参数at指定画刻度线的位置坐标

box()在当前的图上加上边框

rug(x)在x−轴上用短线画出x数据的位置

locator(n, type=”n”, …)在用户用鼠标在图上点击n次后返回n次点击的坐标(x, y);并可以在点击处绘制符号(type=”p”时)或连线(type=”l”时),缺省情况下不画符号或连线

下面的例子中,使用plot(...,type=”n”)绘制一个“空白”的图形, 然后用低级函数来添加点,坐标轴,标签等。■

本文未完,更多内容请参考原文:
http://developer.51cto.com/art/201305/393439.htm

R语言对图片背景透明处理

我们在准备会议或工作报告时经常要把单位或组织的徽标插入到幻灯片中，但从照片或网上截屏获得的徽标图片的背景色通常和我们幻灯片的背景不搭配，能把徽标本身的背景色透明掉就好了。如果你会用R，过程就非常简单，只要几行代码。

JPG/JPEG图像背景是不能透明的，但PNG图像可以。代码如下，应用前提是背景和徽标色能较好地区分（稍修改也可以做背景色替换）：

```
library(jpeg)
library(png)
FILTER <- matrix(c("JPG file", "*.jpg", "JPEG file", "*.jpeg"),  ncol = 2, byrow = T)
img <- choose.files(caption = "Select JPG files", filters = FILTER, index = 1, multi=T)
for(i in 1:length(img)){
  x <- readJPEG(img[i])
  dimx <- dim(x)
  n <- dimx[1]*dimx[2]
  r <- x[1:n]
  g <- x[(n+1):(2*n)]
  b <- x[(2*n+1):(3*n)]
  ps <- 5; ps <- dimx[1]*(ps-1) + ps  # 背景取值，ps为左上到右下角的像素，5。按情况修改
  tv <-  0.1                  # tv为容差范围，0-1取值，越小越精确
  sel <- abs(r-r[ps])<tv & abs(g-g[ps])<tv & abs(b-b[ps])<tv
  alpha <- rep(1, n)
  alpha[sel] <- 0
  x <- array(c(x, alpha), dim=c(dimx[1:2], 4))
  writePNG(x, gsub("(.+\\.)(jpg|jpeg)$","\\1png", img[i], ignore.case = TRUE))
}
winDialog("ok", "Work done!")
```

下面是几张网站徽标截图进行透明处理后在幻灯片中的叠加效果：

在过去的十年中，我已经遇到过数不清的数据再加工任务。无论是把古老的数据库迁移到现代化数据库当中，还是将大规模数据集利用新型处理工具进行牵引等，将数据由一种格式转化为另一种新格式的任务始终不断出现、频度极高甚至每天都有。而且对于包括IT部门在内的绝大多数人来说，这一转换过程仍然神奇乃至神秘。

数据迁移：高难度带来高回报

请允许我花几分钟向大家解释整个转换处理方式，至少是在宏观上形成概念。在阅读之后，大家可能会发现这方面知识有助于帮助各位向非技术人士详尽阐释相关的后端流程。

一切从Excel开始

让我们先从最困难的常见情况：可怕的Excel电子表格开始。就在前一阵子，某家远在他方的企业打算对有价值的业务流程数据进行一番收集与整理，其中包括库存、销售额、客户资料等等。由于缺乏合适的工具，员工制作了一份Excel电子表格，并利用它承载信息。随着时间的推移，其中逐渐积累起数千条记录、电子表格本身的功能性缺失也开始暴露出来。最终，企业管理者决定将这部分数据转换成到真正的数据库当中。不知道他们当时是选择聘请咨询服务公司还是利用内部资源加以处理，总之有人接手了这份工作。

这项任务的第一步是检查数据本身的纯洁度。在理想状态下，电子表格本身会以数据库的方式对信息进行分类，且每个数据填充区都处于对应的列之下——例如姓氏、名字、街道、城市等等。然而事情并不总能如此顺利，有时候两类包含信息交集的单元格会自上而下位于同一列当中。就以联系人列为例，其中很可能囊括了全名、公司、地址、电话号码等多种独立单元。紧随其后的下一列也可能是对方最新订购信息或2012年采购总额等数据，这就给转换工作带来了极大的挑战。

让我们先看看第一种情况——即理想情况，这是最易于处理的状态。数据比较单纯且具备良好的排序，我们可以通过CSV将其导出并通过自定义解析器把它翻译成数据库内容。优秀的CSV解析器能够将所有这些记录逐一抽离出来汇总成整体数组，然后插入到新的数据库当中。在整个过程中，我们可以对数据进行检查及修改，进而使数据格式更好地适应新数据库的要求。

举例来说，我们可能会利用正则表达式处理电话号码信息，旨在将不同格式的电话号码转化为同一标准。这要求大家在将内容插入新数据库之前，调整好表格中包含的一切特殊符号并对字符串进行格式化处理。操作将把（212）555-1212、212-555-1212、2125551212、212555 1212或者212.555.1212等数字转化成统一的电话号码格式（2120555-1212，这既能提高数字的可读性、又便于未来进行搜索。

我们会利用/[^0-9]+/ 这样的正则表达式实现去格式化，然后通过/([0-9]{3})([0-9]{3})([0-9]{4})/表达式将内容重新加工为新的格式，这样最终结果才能正确处理组成分段号码信息的212、555与1212三部分。只要愿意，我们马上就可以对电话号码进行二次格式化。但请注意，如果我们遇到那些因为太长或太短而不可能是电话号码的数字对象时，也要想好对应的解决方式。

让自由格式贯穿始终

然而一旦表格的格式更加自由，处理的难度也就随之提升。地址就是其中最不容易打理的类型，因为它们的格式比较随意、可能需要通过多种不同方式进行格式化。另外大家还需要留意形态万千的街道与城市名称。举例来说，我们必须确保自己能正确识别“华盛顿哥伦比亚特区”、“华盛顿特区”和“华盛顿”这三种缩写（分别为Washington,DC、Washington, DC以及Washington DC），并弄清楚”Winston-Salem, NC,” “King of Prussia, PA,” “Scranton, Penn.,” “N. Providence RI,” “Houston, tx,” and “O’Fallon, IL.”等令人头痛的城市名称与所在州名称组合。

如果不对内容进行严格界定，这些层出不穷的变化完全可能把解析器逼到崩溃，因为我们无法去除其中存在的特殊字符。另外，我们也不能指望通过不同数量的空格来区分城市名称、州名称、州名称缩写或者地名大写等。因此，我们需要创建一套条件表达式，以最大程度确定这些数据所对应的实际城市与国家，甚至必须与包含美国所有城市与州名称的标准数据库进行比照。根据检测结果，我们可能需要对记录进行重新清查或者至少将对应记录加以标记，以备日后手动检查。

到这里，我们才仅仅剥开问题的表面。光是搞清楚每条记录中城市、州名与电话号码就需要我们投入大量资金与精力，更不要说重新加工并重复电子表格中的其它文本信息了——具体工作量取决于其实际内容。

为什么会出现如此被动的局面？这是因为数据项使用了不受约束的自由格式，而它几乎困扰着世界各地的每一家企业。需要强调，Excel并不是这一切问题的罪魁祸首。Access、自主开发的数据库或者其它任何应用都可能引发此类难题。可以说，除非我们严格把住输入数据的类型与格式关，否则数据积累的结果只会是一团乱麻。很显然，解决问题的关键在于建立一套合适的数据库前端来处理数据输入：我们可以在数据进入的同时对其进行清理与调整，这将大大提高数据的准确性与可用性。准确性与可用性，这是优先使用数据库处理信息的主要优势之一。

然而我们也不能忽视利用后端处理这些数据库集类型的努力。目前我们已经拥有各类相当成熟的工具来简化这一流程，但它们仍无法在每个用例中都切实生效。尽管它们能在一定程度上让输入数据保持“标准”，但漏掉的部分同样会引发问题，这种冲突甚至比不进行预处理来得更加严重。

这类工作相当乏味，需要技术人员将全部精力集中在细节之上。它要求我们投入大量人工进行数据检验、试运行、调整并对部分项目开发工作加以前瞻性考量。但总体而言，这一切付出几乎都能获得令人满意的高额回报。

纯洁清爽的数据令日常工作变得极富效率，这也是每家企业所追求的目标。但大家也千万不能低估整个转换过程中可能出现的严峻挑战。■

关于大项目的几点诉苦

文章是一位一线技术人员对所谓大项目的诉苦。包括框架的混乱，基本一个项目一个框架，没有成熟的框架可以复用。框架没有很好的支持多表查询，框架的底层报错机制不好,动不动就报未将对象引用的错误等等。

几年的工作中,经历了2个几十号人以上的大项目.深深体会了,一个好的框架对项目的成成败是多么重要的. 尤其是我上一个项目.做的是一个国内顶尖的医疗公司的一个门户项目.当时由于项目的时间比较紧,没有过多时间去考虑和研究框架.于是就简单引进公司的另外一个框架,到最后的2年多使用时间,就逐渐感觉到了那框架的弊端.到后面项目中的很多同事都反映,该框架不但没有提高效率,而且严重阻碍项目的进度.结果也恰恰证明了这一点.使得我们中的很多开发进度,都是严重推迟了.

当然,一个项目的成败有很多因素.因为我是搞技术的,我想我还是分析一下技术方面的原因吧:

1,一开始时定项目时,由于时间的因素.没有过多的考虑和研究框架.这也是我感觉到的普遍一个项目类型公司的悲哀.一开始为了拿项目, 过多答应客户要求.其实自己并没有那么大的实力去做那事情.当问题出现了,想的解决方法往往都是一些短期的解决方案.以致导致很多该做的事情没做,俗称”欠债”.我们当时引框架也是,由于前面做的工作有点延误了,所以为了给客户交付一个漂亮的项目进度报表.于是把这么重要的框架选型时间也压缩了.而且我们当时公司的框架也比较乱.基本一个项目一个框架.没有成熟的框架。

所以当我们引了别的项目的框架,也没有过多去验证该框架.结果到最后才发现那框架根本适合我们的需求.然而发现了问题,本该要及时纠正. 但由于自己公司,不可能去承担这更改框架的成本,而且客户也不可能去承担.所以到后面就将错就错,不断去用错的框架去做.这样也导致后面的问题越来越多,到最后只能让一些技术好的同事,就当救火队员,采取头疼医头,脚疼医脚的方式去解决当前问题. 可是纸终究是包不住火的,当问题不断出现后,项目经理顶不住了,就只好换项目经理的方式去解决了. 还好,到最后客户也实在无法忍受了,终于愿意自己掏钱成立一个架构优化组,专门去处理相关的架构问题了.

2.我们当时的项目的主要技术问题有,

一,框架没有很好的支持多表查询.

二.框架的底层报错机制不好,动不动就报未将对象引用的错误.或者一些看不懂的错误提示,导致开发人员要通过调试才能找到问题的原因.

三.底层架构,存在很多性能低下,不稳定的代码.

3.我们没有很好的执行当前定下的开发规范.一开始有做coder review工作.可是做了几次后,就再也没去做了.这样导致后面的开发很乱.很多人为了贪求方便,写代码都copy 粘贴的方式.而且的代码的耦合度非常高,经常改一个问题,又会带来了其他问题.而且同样一个问题,有的地方改好了,有的地方又没改好.

4.技术好的人,往往是用来做救护队的队员.没有充分发挥好他们的作用.其实我们做的工作更多是要有前瞻性.我们要把问题,扼杀在摇篮里。■

为什么有些编程语言会死而有些能活下来？

■ 函数式语言和框架依赖运行时来控制日常编码细节诸如迭代遍历，并发执行和状态变迁，但这并不意味着你失去了对这些细节的控制权，在需要的时候你依然可以决定……

在这个系列的第一期文章里，我先是讨论了函数式编程的一些特性，并用Java和函数式的语言展示了这些特性。在本篇文章里，我将继续讨论这些概念如第一级函数，优化器和闭包。但这期文章的主题是控制：即什么时候用它，什么时候需要它和什么时候应该放弃它。

第一级函数和控制

列表1是上次使用Function Java库实现的数字归类器，它拥有isFactor()和factorsOf() 方法：

列表1函数化的数字归类器

```
importfj.F;
importfj.data.List;
import static fj.data.List.range;
import static fj.function.Integers.add;
import static java.lang.Math.round;
import static java.lang.Math.sqrt;
public class FNumberClassifier {
publicbooleanisFactor(int number, int
  potential_factor) {
return number % potential_factor == 0;
   }
public List<Integer>factorsOf(final int
  number) {
return range(1, number+1).filter(new
  F<Integer, Boolean>() {
public Boolean f(final Integer i) {
return number % i == 0;
        }
      });
   }
public int sum(List<Integer> factors) {
returnfactors.foldLeft(fj.function.Integers.
  add, 0);
   }
publicbooleanisPerfect(int number) {
return sum(factorsOf(number)) - number
  == number;
   }
publicbooleanisAbundant(int number) {
return sum(factorsOf(number)) - number >
  number;
   }
publicbooleanisDeficiend(int number) {
return sum(factorsOf(number)) - number <
  number;
   }
}
```

备份的验证最简单的就是找一个空闲的库，来恢复出来，mysql启动以后检查部分数据。如果不需要这么严谨，对于xtrabackup来说，你至少得验证它

apply-log能够恢复上去吧？同样，备库的数据一致性也需要经常检查一下，mysql的replication并不保证100%的数据一致性，你可以去翻翻mysql statement复制的bug列表，有些数据在主备不同的环境上分别执行，数据就会不一样。可以考虑用percona的工具pt-table-checksum来检查主备不一致，用pt-table-sync来同步主备数据。

第5条，对生产环境存有敬畏之心。这应该是运维者进入行业首先需要具备的素质。但是我们还是需要把它拿出来强调一下。

有机会的话，你可以梳理一下：

你的生产环境上有哪些账户，这些账户是否都确实需要登录到机器上来？这些账户即包括linux用户还包括数据库账户。

你的root用户是否开放给了某些用户，这些用户安全吗？

你的用户密码是否经常修改，是否加密不让具体的操作人员直接看到，密码强度时候足够，密码重试次数达到一定次数是否黑名单；

你的生产环境和线下环境是否隔离，数据库是否和外网隔离？

是否一些工作明明能够在开发库和测试库做，却被放到生产环境上去了。

是否有专门的人负责线上应用的发布，从而避免开发人员直接接触生产环境。

这些都是你避免出现csdn密码泄漏，在业界的名声一落千丈的法宝。

第6条，交接和休假最容易出故障，变更请谨慎。这个是经验之谈。

我们在总结故障的情况时，发现在公司部门有变化时，工作交接(不管是休假，工作职责变化还是离职)，故障的出现频率会比正常情况下多50%以上。有人说，这是因为机器或者应用是有感情的，舍不得离开的运维者。

我们不谈感情，简单的理性分析一下。公司或者部门难免会做一些调整，变化是世界上唯一不变的事情。而运维人员是一线做事情的人，部门调整或者领导的更换可能导致工作的着重点不同，做事的方式和评测的标准变了，适应过程中难免会出现一些考虑不周到的地方，出故障也是情理之中了。

而工作交接，对运维人来说，其实是一个非常费时费力的事情，你需要把所有平常做的工作都梳理清楚，甚至包括你的一些经意不经意的操作习惯，这样的话，下一个人才可能接手的下来。比如：你可能认为备库正常情况下没有访问，于是让某些并不重要的任务(一个月一次抽取部分数据到线下测试？)直接连备机IP进行操作。下一个人接手，认为备机就是备机，操作起来不会有任何问题，结果下一次任务抽取就是一个故障出来了。再举一个我们遇到了事例吧：同事A出国休假了，休假期间估计联系不上，他留了文档，并告诫说某几个库和表是比较核心和容易出问题的，没有特殊情况最好等他回来再做变更。正好，休假期间，开发人员找到同事B，要求他重置一个字段的某一位（bit），并打包票说这个bit没有用，同事B拒绝，并背上了不配合的骂名。同事A回来吓了一身冷汗，原来这个字段已经被另外一个离职的开发使用了。

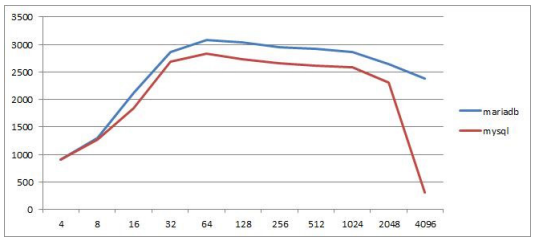
所以，运维部门和运维人员对变化需要尽量放平心态；接手别人的工作要一而再，再而三的

MariaDB 5.5在Windows下性能测试

我们有很长时间没有发布过在Windows下的基准测试文章了，而Windows下的MariaDB包含一些专门做的改进，这些改进很多人并不知晓，因为我们自己也很少提及。本文的目的是向你展示MariaDB在Windows下的性能表现。

进行测试的机器包含2个CPU共8核的处理器（这是我手头上能找到最好的机器了）、10K SAS 磁盘（RAID1），使用sysbench 0.4测试单表共100万行记录。我通过网络来运行这个压力测试，并发的客户端从4到4096。

这里是 OLTP-只读 吞吐量测试结果：



说明：

绝大多数的测试结果表明，MariaDB 的吞吐量比 MySQL 高出 10% 左右

在 4096 并发客户端时，MariaDB 的吞吐量是 MySQL 5.5 的 四倍多 476% (2382 vs 413 TPS).

很多人理所当然的认为吞吐量并不能代表数据库的整体性能表现。在OLTP的基准测试中，响应时间同样很重要，实际上它比吞吐量更加重要，这点我同意，因此，下面是查询的响应时间，意味着 95% 的事务都在指定的时间内处理完毕。

OLTP 只读响应时间

并发数	4	8	16	32	64	128	256	512	1024	2048	4096
MariaDB	4.87	6.81	8.83	12.35	22.12	43.56	90.35	180.57	619.05	1003.88	1965.77
MySQL	4.86	7.14	9.96	16.21	37.39	101.33	238.89	499.63	971.07	2241.83	25215.29

上表中显示，MariaDB 5.5 不管是在吞吐量还是响应时间方面都是优于 MySQL 的。

但是，为什么MariaDB在Windows 下的只读测试由于MySQL 5.5呢？二者基于同一个代码，表现应该也相同啊。这个问题的答案并不是 MariaDB 做了什么优化，也无关 XtraDB 和 InnoDB 的优劣。答案是 MariaDB threadpool. 这个线程池在 Windows 平台是默认启用的。

可是，为什么使用线程池就可以有如此好的性能呢？答案是 MariaDB 承担了通过调整线程池的大小并回调到对应的 Windows 本身的线程池，这在操作系统这一级别上相当于黑盒排序，因此能获取良好的性能。Windows 内置的线程池的核心，是自 NT 3.5 就有的技术，这是 Windows 专有的特性，运行在其上的服务器应该使用这种技术。要让这项技术运行良好的招数是：

不要让同一时间在同一个CPU上运行太多的线程，这样可减少上下文切换，这是提高吞吐量的最重要的因素

在完成的LIFO顺序中激活线程等待，热门的线程保持热门，可降低缓存失效

顺序处理 IO 完成，这是响应时间表现良好因素

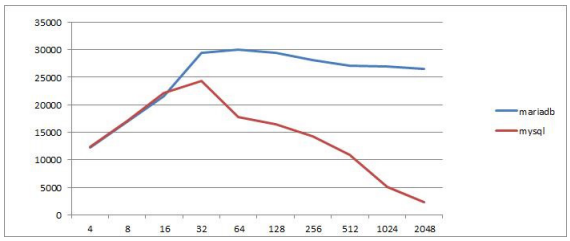
最后便是降低热锁的争用

由此，线程池是只读性能表现佳的主要因素。

下一个有趣的问题是在写操作上MariaDB表现是否一致。因此我们使用写模式来运行 sysbench 工具，也就是 update_non_index (每个查询对一个非索引的整数字段进行加值处理)。为了最大化写的吞吐量，我们设置了参数 innodb_flush_log_at_trx_commit 值为 0，每次日志的写入是每秒一次，而不是每次事务提交一次。

测试结果如下：

OLTP write-only (update_non_index/flush_log=0) 吞吐量:



这个结果看起来很棒，差别来源于多个因素，包括 XtraDB 的写性能、分组提交、线程池等都对这个结果会有影响。但我想Windows平台下的 MariaDB 的 asynchronous IO optimization (异步 IO 优化) 是最主要的因素。

在上述测试中，所有 IO 相关的参数和 InnoDB 参数都使用的是默认值，结果看起来太好了以至于让我们怀疑这是真的。我真的想通过调整为 innodb_io_capacity and/or innodb_write_io_threads 参数为 MySQL 带来更加的性能，有人知道该如何调整吗?

OLTP writeonly (update_non_index/flush_log=0) 响应时间, 95 percentile

并发数	4	8	16	32	64	128	256	512	1024	2048
MariaDB	0.33	0.63	0.75	1.06	1.94	3.85	8.25	21.38	129.79	274.40
MySQL	0.32	0.61	0.73	1.61	7.62	26.82	96.45	219.29	661.19	2723.36

下面是对数据库的参数调整：

```
[mysqld]
sql-mode="NO_ENGINE_SUBSTITUTION"
back_log=500
user=root
port=3306
max_connections=4096
max_connect_errors=5000
max_prepared_stmt_count=50000
table_cache=2048
transaction_isolation=REPEATABLE-READ
loose-skip-external-locking
innodb_status_file=0
innodb_data_file_
  path=ibdata1:200M:autoextend
innodb_buffer_pool_size=1G
innodb_additional_mem_pool_size=20M
innodb_log_file_size=650M
innodb_log_buffer_size=100M
innodb_support_xa=0
innodb_doublewrite=0
innodb_flush_log_at_trx_commit=0
query-cache-size=0
query-cache-type=0
symbolic-links=0
skip-grant-tables
```

■

从程序员到项目经理：不要试图和下属做朋友

■ 谷在项目团队经常有一些比较能干的员工，为项目经理排忧解难，因此渐渐得到项目经理器重。由于互相依赖，两者很容易发展成为朋友关系，有的项目经理甚至将员工当作“心腹”看待，借此来笼络员工，这其实是一种很不明智的做法。

在项目团队经常有一些比较能干的员工，为项目经理排忧解难，因此渐渐得到项目经理器重。由于互相依赖，两者很容易发展成为朋友关系，有的项目经理甚至将员工当作“心腹”看待，借此来笼络员工，这其实是一种很不明智的做法。

从广义上来说，同事也是朋友，同事之间也是存在友情的。在正常情况下，项目经理与每个人的距离是相等的，整个团队保持一种平衡。如果项目经理与某位员工建立了过于亲密的朋友关系，这种平衡将会被打破，从而影响整个团队的凝聚力。

1.得不偿失的朋友关系

表面看上去，工作与朋友并不矛盾。人非草木，孰能无情？长期工作，在同事之间产生友谊，这是可以理解的，而且也有助于建设凝胶型团队。但对于一个管理者而言，如果表现出与下属有亲密的朋友关系，那就实在不妥了。在项目中，朋友关系不但会大大折损项目经理的威信，而且对整个项目团队还会产生许多其它负面的影响。

（1）无法客观公正

一旦下属成为朋友，项目经理在工作就难以像以前一样做到公正客观。毕竟在公司工作主要是讲原则，而朋友之间则要讲感情，如果朋友这间处处以真理为依据，以大是大非为准则，这样的朋友估计是做不长的。同样如果工作中带进过多的感情色彩，工作也会变得难以开展。

我们不妨设想一下，和员工成为朋友这后，碰到下面这些情况你会如何处理：

●朋友在公开场合发表不恰当的言论，或者打乱项目内的等级秩序和工作流程，你会像批评其他人一样直接批评他吗？还是用对方可能听不懂的话进行暗示提醒？

●在对员工进行绩效考核时，你会不会因为感情原因，不自觉的拔高一些他的分数呢？

●在其任务不能保质保量、按时完成时，你的要求是不是也降低或更富有弹性了？

●你是不是不能像以前那样自然的来检查他的工作了？

●在检查他的工作时，发现了技术问题，你是不是也不好意思进行指导了？

●在他变得自以为是、经常对你提一些不切实际的建议和要求时，你是不是也无可奈何了？

一边是原则，一边是友情，该如何抉择？这就是把朋友关系带进工作后，项目经理面临的困境。面对上面的这些问题，项目经理要想持守中道，就必须摒弃感情的因素，以原则为导向，以事实为依据，做出冷静的选择。这样，不管将来项目中会出现什么问题，项目经理都可以做到问心无愧，进退有据。

（2）给所有成员带来错觉和困扰

工作中的朋友关系不只是给双方带来不便，而且会给其他员工带来错觉和困扰，从而影响项目的凝聚力和战斗力。

在IBM日本总部曾发生过一个著名的“东京事件”：

IBM“东京事件”的起因，是IBM东京公司高层决定秘密重奖几位工作出色的骨干分子。这件事本来是机密，在美国IBM本部也是一种例行的激励手段，但让管理层意想不到的是，领奖的几个人刚走不久，一些没有得到奖励的人就跑来要求辞职。他们这么做倒不是出于闹情绪，原因很简单——别人被重奖，而自己没有得到奖励，证明自己工作成绩不突出，得不到领导认可，继续“混”下去没劲，还不如自己知趣点，主动申请走人，免得日后被老板裁掉那么尴尬。令管理层更想不到的是，等这些人刚走，那些受到奖励的人又跑来要求辞职！原因更简单——由于自己被老板重奖的原因，害得同事们丢了饭碗；而同事因此辞职又害得公司工作陷入了被动。所以是既对不起同事也对不起公司，只好坚决辞职，以谢同事和公司。

这个事件看起来很诡异，对骨干员工的奖励居然会导致所有员工辞职，但这件事同时也是可以理解的，因为对个别的秘密奖励破坏了员工之间原有的平衡关系。这件事也让我对日本人的团队精神刮目相看，我想这也是我们该好好学习的地方。

虽然故事中是对部分员工进行物质奖励，与我们谈的朋友关系似乎没有什么联系。但两者对团队和谐的破坏是相同的，我们完全可以进行类比。项目经理和下属的朋友关系，在一定程度上，就好比是对个别员工的特殊奖励。其他员工会想，“既然经理跟他这么亲近，对我们这么疏远，想必我们没有什么价值”，这样团队的士气必然大打折扣。项目经理的同事朋友回头一想，也许会觉得“经理对我一个人这么好，肯定会引起其他人的不满，我还是离经理远一点才好”。这样一来，整个团队都会陷入不必要的困扰中。

一个和谐团队内部，员工之间会保持一种微妙的平衡，它源自项目组成员之间彼此平等、互相尊重的关系，以及相互之间的乐于接受的评价和看法。一旦组织内部出现某种特殊关系，这种平衡就会遭到破坏。

在项目中如果项目经理与个别员工建立亲密的朋友关系，这对其他人的思想观念会产生很大的冲击，搞不好就会其他人“三观尽毁”：

● 对自己的看法

他们会想，是不是经理认为我能力差？我在团队是不是不重要？在考核或分配奖金时项目经理会不会也厚此薄彼？在被批评时，会想经理是不是有意对我刻薄？不行，看来没有前途，要走人了！

● 对项目经理“朋友”的看法

那个家伙编程不怎么样嘛，有问题还不是问我？只会花言巧语，博得经理高兴。

● 对项目经理看法

这个项目经理不怎么样，没有威信，喜欢听好话，跟着他干没前途。

也许那个下属确实能力超群，也许项目经理能够尽力把握公正与平衡，但这些不足以挽回项目经理因表面上“偏心”给团队带来在伤害。

2.正确认识员工与公司的关系

为什么项目经理难做？一个重要原因，就是项目经理具有双重身份。当面对员工和客户，他代表公司；面对老板，他又代表员工。因此项目经理经常需要处理公司与个人之间的一些问题，一个经验丰富的项目经理，也必然更加懂得公司与员工之间的关系。

很多公司为了提高凝聚力，宣称“员工是主人翁”、“公司是大家庭”等等，这得到无数人的认可。既然公司是家庭，那员工也就是家庭成员了，这样看起来员工与公司的关系应该非常亲密，员工之间也应该如同兄弟姐妹一般才对，那项目经理与员工怎么连朋友都做不得了呢？

写到这里，不由得想起了在2004年联想公司的裁员风暴中，曾有一篇流传广泛的文章叫《公司不是家》。作者目睹了曾经一起为梦想奋斗、以联想为家的同事，几天之内一批批被遣散。被裁的员工事先都完全不知情，在面谈之前，他们的一切手续公司都已经办完，邮箱、人力地图、IC卡全部被注销，当他们知道消息以后，两个小时之内必须离开公司。

作者在文中伤感的写道：“我突然想起来二战时某位著名将军说的话：“我让士兵上战场的时候，我会把他们想象成一堆蚂蚁，而不是人。因为我一想到他们有妻子、孩子、父母，我就不忍心让他们去送死。”不知道领导在讨论名单的时候，是把我们想象成蚂蚁吗？……我想，我比许多人都体会深刻。员工和公司的关系，就是利益关系，千万不要把公司当成家。”

联想董事长柳传志也对这篇文章做出了回应，他说：“我很抱歉地对《公司不是家》的作者说，我们考虑问题的角度不同。元庆只能从企业发展的角度，从大局的角度看问题，这才是最根本的以人为本，最根本的为员工负责。如果元庆真的用为局部员工负责的方法去考虑问题，企业就会陷入一片儿女情长之中，完全无法发展，中国就会失去联想。因此企业前进的主旋律只能是战鼓，是激昂。”

大裁员是一件很惨烈的事情，但这不能怪公司，它的生存法则决定了它只能这么做。

通过联想这位员工与老板的对话，其实已经清楚的把员工与公司的关系说出来了，其实公司根本不是什么家，只是工作的地方而已，公司出钱请员工干活，就这么简单。柳传志说裁员是为了更好的以人为本，如果公司是家的话，那这就好比家长对孩子说：“为了让全家人都有饭吃，我只好把你仍掉了”，岂不荒谬？所以那些号称“公司是大家庭”的老板们，如果你们做不到永不裁员、永远要给员工生活的保障，那还是请你们自行撤下这虚伪的面具吧，因为没有哪个家庭会抛弃自己的兄弟和子女。作为员工，也必须清醒的认识到，你和公司之间就是一种利益关系，你是为自己工作，绝不是为了公司这个“家”。你之所以在这里工作，是双方利益的需要，绝不是感情的原因。柳传志所说的“考虑问题的角度不同”，其实质只是利益不同而已。

话已经说得很白了，看上去有点残酷。有些人觉得伤感，好像自己对公司的感情被一棒子打入冰窖，就好像一个活生生的人，突然失去了血肉、变成了骷髅一样。其实大可不必这么想。员工与公司有其相处的模式，只不过这种模式绝不是家庭模式，也不是朋友模式。我们应该坦然面对，细心揣摩，谨慎把握。■

本文未完，更多内容请看原文：
http://developer.51cto.com/art/201305/392755.htm

Java 8的新特性和改进总览

□开源中国编译

这篇文章是对Java 8中即将到来的改进做一个面向开发者的综合性的总结，JDK的这一特性将会在2013年9月份发布。

在写这篇文章的时候，Java 8的开发工作仍然在紧张有序的进行中，语言特新和API仍然有可能改变，我会尽我最大的努力保持这份文档跟得到Java 8的改动。

Java 8的预览版，也就是 “Project Lambda”，现在可以从java.net下载到。

我使用了IntelliJ的预览版做我的IDE，在我看来他是目前支持java 8特性最好的一个IDE，你可以从这里下载到.

由于我没有找到Oracle发布的Java 8的官方文档，所以目前Java 8的文档还只有本地版本，等Oracle公开文档的时候，我将会重新链接到官方文档。

接口改善

现在接口里已经完全可以定义静态方法了. 举一个比较普遍的例子就是在java类库中, 对于一些接口如Foo, 都会有一个有静态方法的工具类Foos 来生成或者配合Foo对象实例来使用. 既然静态方法可以存在于接口当中, 那么大多数情况下 Foos工具类完全可以使用接口中的公共方法来代理 (或者将Foos置成package-private).

除此之外更重要的就是, Java 8中接口可以定义默认的方法了.举个例子,一个for-each循环的方法就可以加入到java.lang.Iterable中:

```
public default void forEach(Consumer<?
  super T> action) {
      Objects.requireNonNull(action); for (T
  t : this) {
            action.accept(t);
      }
}
```

在过去,java类库的接口中添加方法基本上是不可能的. 在接口中添加方法意味着破坏了实现了这个接口的代码. 但是现在, 只要能够提供一个正确明智的默认的方法的实现, java类库的维护者就可以在接口中添加方法.

Java 8中, 大量的默认方法已经被添加到核心的JDK接口中了. 稍候我会详细介绍它们.

为什么不能用默认方法来重载equals，hashCode和toString?

接口不能提供对Object类的任何方法的默认实现。特别是，这意味着从接口里不能提供对equals，hashCode或toString的默认实现。

这刚看起来挺奇怪的，但考虑到一些接口实际上是在文档里定义他们的equals行为的。List接口就是一个例子了。因此，为什么不允许这样呢?

Brian Goetz在这个问题上的冗长的回复里给出了

4个原因。我这里只说其中一个，因为那个已经足够说服我了：

它会变得更困难来推导什么时候该调用默认的方法。现在它变得很简单了：如果一个类实现了一个方法，那总是优先于默认的实现的。一旦所有接口的实例都是Object的子类，所有接口实例都已经有对equals/hashCode/toString的非默认实现。因此，一个在接口上这些的默认版本都是没用的，它也不会被编译。

要看更多的话，看下由Brian Goetz写的解释：对“允许默认方法来重载Object的方法”的回复

函数式接口

Java 8 引入的一个核心概念是函数式接口。如果一个接口定义个唯一一个抽象方法，那么这个接口就成为函数式接口。比如，java.lang.Runnable就是一个函数式接口，因为它只顶一个一个抽象方法：

```
public abstract void run();
```

留意到“abstract”修饰词在这里是隐含的，因为这个方法缺少方法体。为了表示一个函数式接口，并非想这段代码一样一定需要“abstract”关键字。

默认方法不是abstract的，所以一个函数式接口里可以定义任意多的默认方法，这取决于你。

同时，引入了一个新的Annotation：@FunctionalInterface。可以把他它放在一个接口前，表示这个接口是一个函数式接口。加上它的接口不会被编译，除非你设法把它变成一个函数式接口。它有点像@Override，都是声明了一种使用意图，避免你把它用错。

Lambdas

一个函数式接口非常有价值的属性就是他们能够用lambdas来实例化。这里有一些lambdas的例子：

左边是指定类型的逗号分割的输入列表，右边是带有return的代码块：

```
(int x, int y) -> { return x + y; }
```

左边是推导类型的逗号分割的输入列表，右边是返回值：

```
(x, y) -> x + y
```

左边是推导类型的单一参数，右边是一个返回值：

```
x -> x * x
```

左边没有输入 (官方名称：“burger arrow”)，在右边返回一个值：

```
() -> x
```

左边是推导类型的单一参数，右边是没返回值的代码块（返回void）：

```
x -> { System.out.println(x); }
```

静态方法引用：

```
String::valueOf
```

非静态方法引用：

```
Object::toString
```

继承的函数引用：

```
x::toString
```

构造函数引用：

```
ArrayList::new
```

你可以想出一些函数引用格式作为其他lambda

方法引用	等价的lambda表达式	
String::valueOf		x -> String.valueOf(x)
Object::toString		x -> x.toString()
x::toString		() -> x.toString()
ArrayList::new		() -> new ArrayList<>()

当然，在Java里方法能被重载。类可以有多个同名但不同参数的方法。这同样对构造方法有效。ArrayList::new能够指向它的3个构造方法中任何一个。决定使用哪个方法是根据在使用的函数式接口。

一个lambda和给定的函数式接口在“外型”匹配的时候兼容。通过“外型”，我指向输入、输出的类型和声明检查异常。

给出两个具体有效的例子：

```
Comparator<String> c = (a, b) -> Integer.
  compare(a.length(),
                        b.length());
```

一个Comparator<String>的compare方法需要输入两个阐述，然后返回一个int。这和lambda右侧的一致，因此这个任务是有效的。

```
Runnable r = () -> { System.out.
  println(“Running!”); }
```

一个Runnable的run方法不需要参数也不会返回值。这和lambda右侧一致，所以任务有效。

在抽象方法的签名里的受检查异常（如果存在）也很重要。如果函数式接口在它的签名里声明了异常，lambda只能抛出受检查异常。

捕获和非捕获的Lambda表达式

当Lambda表达式访问一个定义在Lambda表达式体外的非静态变量或者对象时，这个Lambda表达式称为“捕获的”。比如，下面这个lambda表达式捕捉了变量x：

```
int x = 5; return y -> x + y;
```

为了保证这个lambda表达式声明是正确的，被它捕获的变量必须是“有效final”的。所以要么它们需要用final修饰符号标记，要么保证它们在赋值后不能被改变。

Lambda表达式是否是捕获的和性能悄然相关。一个非不捕获的lambda通常比捕获的更高效，虽然这一点没有书面的规范说明（据我所知），而且也不能为了程序的正确性指望它做什么，非捕获的lambda只需要计算一次. 然后每次使用到它都会返回一个唯一的实例。而捕获的lambda表达式每次使用时都需要重新计算一次，而且从目前实现来看，它很像实例化一个匿名内部类的实例。

Lambdas不做的事

你应该记住，有一些lambdas不提供的特性。为了Java 8它们被考虑到了，但是没有被包括进去，由于简化以及时间限制的原因。

Non-final* 变量捕获 – 如果一个变量被赋予新的数值，它将不能被用于lambda之中。”final”关键字不是必需的，但变量必须是“有效final”的（前面讨论过）。这个代码不会被编译：

```
int count = 0;
List<String> strings = Arrays.asList(“a”, “b”,
  “c”);
strings.forEach(s -> {
    count++;
// error: can’t modify the value of count });
```

例外的透明度 – 如果一个已检测的例外可能从lambda内部抛出，功能性的接口也必须声明已检测例外可以被抛出。这种例外不会散布到其包含的方法。这个代码不会被编译：

```
void appendAll
(Iterable<String> values, Appendable out)
  throws IOException {// doesn't help with
  the error values.forEach(s -> {
        out.append(s);
// error: can't throw IOException here //
  Consumer.accept(T) doesn't allow it});
}
```

有绕过这个的办法，你能定义自己的功能性接口，扩展Consumer的同时通过像RuntimeException之类抛出 IOException。我试图用代码写出来，但发现它令人困惑是否值得。

控制流程 (break, early return) –在上面的 forEach 例子中，传统的继续方式有可能通过在lambda之内放置 ” return;” 来实现。但是，没有办法中断循环或者从lambda中通过包含方法的结果返回一个数值。例如：

```
final String secret = "foo";boolean
  containsSecret(Iterable<String> values)
  {
    values.forEach(s -> { if (secret.
  equals(s)) {
          ??? // want to end the loop and
  return true, but can't }
    });
}
```

进一步阅读关于这些问题的资料，看看这篇Brian Goetz写的说明：在 Block<T>中响应“已验证例外”

为什么抽象类不能通过利用lambda实例化

抽象类，哪怕只声明了一个抽象方法，也不能使用lambda来实例化。

下面有两个类 Ordering 和 CacheLoader的例子，都带有一个抽象方法，摘自于Guava 库。那岂不是很高兴能够声明它们的实例，像这样使用lambda表达式?

```
Ordering<String> order = (a, b) -> ...;
  CacheLoader<String, String> loader =
  (key) -> ...;
```

这样做引发的最常见的争论就是会增加阅读lambda的难度。以这种方式实例化一段抽象类将导致隐藏代码的执行：抽象类的构造方法。

另一个原因是，它抛出了lambda表达式可能的优化。在未来，它可能是这种情况，lambda表达式都不会计算到对象实例。放任用户用lambda来声明抽象类将妨碍像这样的优化。

此外，有一个简单地解决方法。事实上，上述两个摘自Guava 库的实例类已经证明了这种方法。增加工厂方法将lambda转换成实例。

```
Ordering<String> order = Ordering.
  from((a, b) -> ...);  CacheLoader<String,
  String> loader = CacheLoader.from((key)
  -> ...);
```

要深入阅读，请参看由 Brian Goetz所做的说明： response to “Allow lambdas to implement abstract classes” 。■

本文未完，更多内容请看原文：
http://developer.51cto.com/art/201305/392308.htm

菜菜是个开朗乐观的90后小文艺少女，随和开放。饭饭是个睿智严谨的80后程序员，温和传统。她还是个大学生，他已是…

中文女和程序员的爱情奇遇

“我所认为最深沉的爱，莫过于分开以后，我将自己，活成了你的樣子”。——写给所有热爱互联网和相信爱情的人。菜菜是个开朗乐观的 90 后小文艺少女，随和开放。饭饭是个睿智严谨的 80 后程序员，温和传统。她还是个大学生，他已是工作族。故事的发生始于青天白日被一大捆 Money 砸中的相爱几率，两个人的生活也从此发生了翻天覆地的变化。

很多人想象中的程序员，应该是呆板的、不修边幅、不懂时尚不会打扮之类的技术宅男。跟饭饭在一起后，菜菜彻底改变了这个偏见。饭饭穿着整洁干净，谈吐得体幽默，但宅男是真，不会打扮也是真。菜菜给饭饭制定了一套全方位 360 度无死角转型方案，她给他分析女生喜欢男生的装扮方式，她教他如何恰当的服装搭配和颜色协调。她打破他一贯的穿衣审美模式，给他挑了粉色的 7 分袖衬衫，淡蓝色的小西装，米白色的针织开衫，V领的毛衣，白色的球鞋和浅绿色的休闲鞋。再拉着他去蓝调做了个潮流发型。一番打造之下，一个时尚又不失温文尔雅的型男像面包一样新鲜出炉！菜菜摸着下巴心满意足地看着揣测不安的饭饭，随之大喝一声，扑上去双手揉捏着饭饭的脸颊媚笑：“亲爱的！你肿么可以这么帅！！”

菜菜是个如假包换的文科女，从小到大饱读诗书，有诸如张小娴、席慕容、简祯的柔软清新文字，有张爱玲、三毛、廖一梅的不羁豪放文风，也有诸如村上春树、渡边淳一、杜拉斯的重口味文学。同是晒太阳，她一张口就能来句“俄罗斯有位诗人曾说，我来到这个世界是为了看看阳光”，随之酸倒一脸愕然的舍友们。中文系的浓墨重彩使她游离于现代科技之外，她是个笨到连电脑、热水器都不会正确使用的技术白痴。直到她遇到了饭饭，一个从事着用伟大的代码改变互联网世界的程序员，一个破天荒说她适合当产品经理的人。

都说程序员的双手是魔术师的双手，他们通过技术把世界变得更加丰富多彩。饭饭就是这样一个无所不能的高级软件工程师。饭饭在菜菜心里不仅是恋人，也是兄长和老师。每次菜菜花痴状地趴在饭饭的膝头上，张着澄澈的闪烁着无限崇拜光芒的大眼睛仰视饭饭写代码，他神一般的盲打在键盘上啪啪的敲击声，工程窗口连续不断地跳跃出一行行的英文字符，那架势堪比迪拜高塔平地崛起的建筑传奇！男人认真的时候性感得要命啊！！饭饭依旧面不改色，他淡定地瞥着一旁作流口水状的菜菜，嘴角扬起 30 度的贼笑：不稀的说你。

于是乎，菜菜开始发愤图强学起跟 IT 有关的知识，她梦想有天能成为饭饭得力的事业助手。她是

个纯电脑白痴，她眼神放空地盯着饭饭书桌上的一大堆 IT 书籍几秒后，她的心里豪迈地升腾起一个宏伟的目标：我要挺进 IT！

饭饭拿给她看的第一本书是《结网：互联网产品经理改变世界》，菜菜二话不说便斗志昂扬专研起来，不论上课下课她都雷打不动地看着。舍友们好奇于她的痴迷，抓来她的书一瞥，随之隔行隔座山地摇摇头并嬉笑她走火入魔。菜菜认真地看完两遍之后，她兴奋地跟饭饭谈论她的心得感受。她跟饭饭侃侃而谈用户体验三要素：别让我等！别让我想！别让我烦！她发微博拿《结网》这本书作调侃，却冷不丁地收到了作者本人的回复！饭饭一听仰天长啸三声，他做了这么多年的技术从未被名人赏识，菜菜这等小毛孩居然踩了狗屎运！

饭饭非常欣慰和惊讶，这个小丫头的领悟能力和学习能力不错啊！孺子可教也！接下来，他继续给菜菜推荐书籍，《写给大家看的设计书》，《Don’t make me think》，《人人都是产品经理》，《定位》，《如何玩转你的网站》，《浪潮之巅》。。。菜菜顶着兼容并包的学习心态打鸡血似的一本本啃完。

饭饭开始给菜菜耐心细致地解释，什么是云计算，PHP, Java, Html，什么是 SEM, SEO, SMM, CPC, PPC 等专业术语，菜菜启动所有大脑神经来者不拒，虽然理解有些困难，但仍然一脸微笑地挑战自己的抗压能力全盘接收。

后来菜菜翅膀硬了开始哼哼鼻子如是说：

“你知道完美的网页设计需要具备哪些原则吗？哇哈哈，让我大发慈悲地告诉你，完美的网页设计需要具备亲密性、对齐、重复和对比 4 个基本原则。不知道了吧？！”

“你知道网页最好的用户体验是什么吗？那就是 Don’t make me think！”

“你知道你的网站为什么会不定期地受到黑客攻击？其实是这样的，这个黑客就是我！只要你惹我不高兴，你的网站就 down 了。害怕了吧？！”

“我觉得你这个功能设计不够好，应该改成鼠标滑动显示的效果。你觉得怎么样？”

“这几行代码写得简洁利落，很好！不过，这个分隔符写错了，应该是；而不是：。”

书看多了后，菜菜开始嚣张叫板技术牛人饭饭，她以一个门外汉的无知无畏在饭饭面前班门弄斧，饭饭每次听到她玩笑的挑衅，神色总会大放异彩，活像培养了一个开窍的弟子自己后继有人一般！饭饭总是不吝言辞夸她聪明，他总是意味深长地说，小丫头以后一定会超过我的。他对她说，IT 公司的工作氛围平等自由，正好适合她无拘无束的脾性，他期望她将来能够进腾讯。菜菜听得眼里跳跃着梦的光影。

菜菜开始学习 Dreamweaver，学习 PS，她能当测试员？或网页设计师？或数据分析师？Anyway，尽管她是只 IT 的蜗牛，她也要坚忍不拔响应 JAY 的歌曲：我要一步一步往上爬。记住记住！菜菜要当饭饭的事业助手事业助手！

菜菜一直梦想着有天能和饭饭一同工作，直到有天饭饭的同事给她提供了一个实习生的工作机会。菜菜就这样莫名其妙地，对自己半信半疑地进入 IT 公司工作。她和饭饭的五个部门伙伴成为了好朋友，也认识了公司很多有趣友善的人，她经常跟饭饭说起很喜欢这些人！直到现在，菜菜仍然深深感激他们给予自己的成长、宽容和帮助。

菜菜的岗位是 SMM 市场推广，她每天厚着脸

皮悉心求教，直至用低智商的问题把旁边的组长问得一脸茫然诧异！当她第一次获得经理的夸赞时，她兴奋地对饭饭一阵炫耀。饭饭高兴之余，总不忘教导她戒骄戒躁，他希望她能有自己的成长空间，他不会一味的宠溺让她自我感觉过于良好。菜菜深谙此理， 她无非是想要得到饭饭的肯定。

饭饭每次带菜菜上街觅食，他总是先拿起手机点击大众点评做参考。他每次看到新软件总会习惯性评论这个产品做得怎么样。菜菜在嬉笑他的职业病之余，她也认真地研究起产品的功能来。她以前只会玩玩美图秀秀和聊天工具，饭饭给她安装了很多使用性的软件后，她俨然也成为一个产品迷。

程序员是个加班户，饭饭更是有过之无不及。菜菜恶狠狠地假装要打电话给饭饭的老总讨加班费。菜菜每次下班都会在楼下的咖啡厅等他，饿得实在忍不住了就给他打电话催他下来，这位淡定哥每次都会说，马上就好，等我 5 分钟。事实上，饭饭每次都是 20 分钟后再下来。有时菜菜会到饭饭的工作部门等他，一脸痴迷地看着他认真工作的模样，安静地只是坐在一旁不去打扰他。再后来，菜菜良心发现想要学厨艺，她从一个不会做饭的马大哈变成了逛菜市场研究厨艺的人。她胡乱忙活终于完成一顿晚餐，但她仍饿着肚子望眼欲穿地等着发布未成功的程序员归来。饭饭心疼地责怪她干嘛不先吃饭。当饭饭终于对她的厨艺夸赞时，菜菜的自豪感和成就感极大地被满足。

菜菜是个安全感缺失的人，她最初害怕一个人待在饭饭的房间。饭饭每天晚上都会提垃圾袋到楼下， 菜菜都会叮嘱他马上回来。有次饭饭倒完垃圾，路过老总的办公室便被叫进去谈工作，一谈就是一个多小时。他没带手机，菜菜恐惧不安地等着他开门进来的那一刻。等到饭饭终于回来时，她趴在他的怀里像个受委屈的孩子一样哭泣。饭饭疼惜地哄着。像孩子依赖着肩膀，菜菜深信不疑只有这个男人才能给予自己全部的安全感。

饭饭是个做事很专注很有责任心的人，一工作起来他常常忘了菜菜交代他办的事情，菜菜会生气地闹脾气但更多的理解他，饭饭喜欢她的善解人意。饭饭不浪漫，他不会制造惊喜，但菜菜认为生活细节上他的体贴和懂得就已足够。

当菜菜和饭饭聊起 CSDN，博客园，51CTO博客，聊起逆向工程，聊起 32 位环境下的汇编...饭饭惊奇地拍案而起，小女　丝逆袭！菜菜也会跟饭饭畅谈文学，每当菜菜一脸鄙视地嘲笑他是二逼青年时，饭饭总会不服气哼哼鼻子瞪大眼睛说，想当年哥也是个文艺青年，还差点写起武侠小说，但走上 IT 的不归路后就此绝笔。菜菜有个作家梦，饭饭鼓励她到盛大文学写写网络小说。她同样激励饭饭说，以后有时间的话可以写写技术类的书籍，或者她来帮他写。

有天菜菜突发奇想参加了北京大学生电影节 DV 创作大赛，她自编自导自演了一部微电影。剧情是关于一个计算机系的男生和一个中文女生的励志爱情故事。为了迎合剧情，她让饭饭做了个虚拟的百度页面和个人网站首页。其实这是菜菜为饭饭设计的心思。

北京到福州距离 2331 公里，但两颗默契的心却近在咫尺。菜菜为了省钱，很多次坐了 20 个小时的硬座去看饭饭，她不让饭饭给她买卧票。异地恋的辛苦和甜蜜也只有自己才懂。

最单纯的情感也有它深不可测的一面。再死去活来的爱也抵挡不过分开命运。■

本文未完，更多内容请看原文：
http://developer.51cto.com/art/201304/390346.htm

失业的程序员：创业就是一场戏

编者按

本系列正式英文系列名是：《the jobless programmer》。简称TJP，又意：碳减排或特价品 。因为程序员是世界上最低碳环保的职业。同时程序员又是在业界无时无刻不在……

本系列正式英文系列名是：《the jobless programmer》。简称TJP，又意：碳减排或特价品 。因为程序员是世界上最低碳环保的职业。同时程序员又是在业界无时无刻不在搞特价或被特价的职业人群。

昨天写到一半死机了，坑爹啊。没有及时的保存，导致后面一大部分得重写。对不起各位了，更新晚了。

一、微前戏：雅安地震

开章前先祝福和祈祷一下雅安人民，祝你们早日度过难关。

前几日玩微博，看到竟然有人发国难财，利用假寻亲电话骗人拨打收费。个人觉得这种人应该拖到马路上用卡车来回碾数次才能解恨。

二、微前戏：商业头脑

励志故事，台湾蜜雅科技公司董事长郑林月宫现年91岁。做了一辈子家庭主妇，80岁开始创业，开发了“蜜雅小烧卖”健康环保电锅系列产品，证明了创新点子就在生活中。

听到这个故事联想到：有一些old版本的程序员(包括我前几年)开始抱怨：30过了，咋办？我还能干什么。于是很多人开始说：转型或转行或创业。然后会有一些人抨击：说转型谈何容易、转行谈何容易、创业更谈何容易。

我放点个人胡扯的小观点（好久不正面说自己的观点，可能超出上面的主题）：

1、任何时候创业都不晚。人家80岁还能开始创业，我们才一半不到，有什么理由迷茫。

2、只要活着，就要抱持梦想活下去。只要抱着希望燃烧热情，我们永远很年轻。（不懂的说明还没到这个年龄或者您已经成功鸟）

3、在我国创业必须源于生活。在国外可以源于精神。

4、程序员创业不一定要源于IT，转行不代表你干的一定不是程序员。转行还是没有转行主要看出发点。很可能你一开始只是为了赚钱或建立“软件帝国”，到后来你用你的技术为社会做出了贡献，这也是一种转型或转行。而成功往往会垂涎后者。

三、微前戏：鸡生蛋还是蛋生鸡

我昨天突然搞懂了这个问题。我原来一直以为我们只有 ”现在“这个阶段，过去的已经过去了，未来的还没到来。所以要好好珍惜现在。如果抱这个态度，那将永远搞不懂鸡生蛋还是蛋生鸡。

其实根本就没有”现在“这个阶段，过去在上一秒，未来在下一秒，所以没有现在只有将来和过去。也就是说，鸡代表过去，那么蛋就是将来；如果蛋代表过去，那么鸡就是将来。很多人搞不懂这个问题是因为把注意力都集中在了”生“这个字，

忽略了蛋和鸡这个最真实的关键要素。

换位思考到我们程序员阶段是一样的。

1、其实 没有”现在这么差、现在收入这么少、现在不被认可、现在很难有出路“这个词语。差代表过去你没积累，将来很难快速变好。收入少说明过去你没打好基础，将来努力了一样能成大神。其他依次类推

2、我们真正能把握住的不是”现在“，而是过去和将来。能把过去总结的好，就能充分把握未来。能把未来创造出来，你的过去才有意义。

以上是 失业的程序员 第九章的微前戏。 希望能和大家共勉 最近我在工作和生活中体悟到的一些微不足道的道理。

以下是正文部分：

一、正文：跨入电商

说到我家人。我父母都是小学教师，对我从小管教无比严厉。他们希望我将来也能子承父业或者母业，并且更希望将来我的子女也能如此继承着。用句行内话，接口 定死了，我想不实现接口那是不可能的，最好的办法就是不继承这个类。于是我大学毕业后选择进入一家小小的软件公司，美其名曰在公司里做培训讲师，其实就是 软件实施外加操作培训人员，在一帮带着孩童般幼稚表情的客户中培训着软件如何操作如何使用。我自己PS了一张“优秀软件讲师”奖状带了回家，我父母无比的 欣慰，说等我以后有了子女也要好好培养”他”或”她”，并且最好是”他”。我无奈，有时我大脑会浮现出“山村老师”和“葫芦娃”的情景。

不过我总算名正言顺的做了一回自己的主。

当我还是菜鸟九段程序员的时候，曾有过一段迷茫像只无头苍蝇的时期。我每天除了吃饭、喝水、藐视地球和调控膀胱等四件不得不做的事情外，最常干的事情就是到网上去咨询IT高手：像我们IT新人人干什么才有前途？。结果高手A告诉我做软件是死路一条，高手B告诉我做互联网意味着倾家荡产，高手C告诉我只有做硬件才能快速来钱，高手D告诉我让我汇款3000到XXX账户他立马教会我怎样成功。

当时经过我反复思量并且和高手D对骂了一个小时后，毅然选择坚持走软件即未来的道路。于是后来碰到了卞工、猪刚烈、女组员。我想说他们都是对我今后的路或多或少产生一定影响的人。

至始至终我都不认为做软件是死路一条，尤其当我把做完的软件热热乎乎的交付用户后，看到用户像小孩子般(可能用婴儿来形容更靠谱)的使用着我的软件时，这种满足感和快感是用粉末状的物体都无法带来的。当然，偶尔也会体会到“小孩子真的很难沟通和教化”这种感觉。

直到前几天，我都在为这种“和小孩子”打交道的快感努力而奔波着。学姐和一位被称为文哥的老板闯入了我的办公室。

离开正式招投标还有一天，我、卞工还有耿工正在做最后的标书梳理和技术问答准备。倒也不是我不相信卞工的临场应变能力，原先在猪刚烈手下时也有一次招标， 由于老猪高估了一个新人，结果当时的技术问答环节遇到了暴强的评委，连问五个很刁难的问题，新人一下子懵了，以至于回答的都有点离谱。于是评委开始用很严 厉的话语质疑公司的实力，结果该厮在现场就哭了，坐在台下的老猪和我当时就疯了。俗话说一朝被评委咬，十年怕评委，这次技术问答环节本打算让卞工试手，最后思来考去，我还是决定自己亲自上，万一卞工临场把持不住崩溃了，那就前功尽弃了。

卞工很失望，我拍着胸脯保证下次再有机会一定让他尽展光辉报效社会。耿工在旁边坏坏的笑着，我瞄了他一眼，心想还有件事等这事完了再跟他“算账”。

其实我很喜欢卞工这种身先士卒的精神，有表现欲并且也有真材实料的员工一定要好好培养使用同时还要加以引导，用得好那他就是一颗金子，用得不好全公司就我是一颗金子。

敲门声。

“学姐？” 卞工猛的跳了起来，仿佛从飞机上弹射出来。我和耿工对卞工强烈的反应一个表示很不满一个表示很诧异。

耿工跑去开门，果然是学姐。我诧异卞工竟然已经能从敲门声灵异的感受到是谁。

学姐无比淡定的和我们打招呼，并仔细打量了一下新员工—耿工。耿工被学姐突如其来的打量致使脸颊起了红晕。我此时好想和耿工互相换个位置。

“这是文总，XXX粮油食品公司的老总”学姐给我们介绍她身后的一位很富态的老板。

我们挨个和文总握手。我估计卞工会比较诧异，我倒是回想起来了。前几天，学姐给我打过一个电话，说是一个企业老总想从实体转向电商行业，这不，我前几天还把方案赶好了。

“别这么客气，你们叫我文哥，老文，都行”文哥很谦虚，我对文哥第一印象不错。很少有老总级的人物能如此平易近人，至少看起来是。

“我们先谈正事吧，等下文总还有其他事”，学姐的高效率是不容忽视的。 我本想客套一番，拿出我藏在抽屉里很长时间的好烟发一下。我认为像文总这样的大老板一定会回敬比我牛叉几倍的好烟。

耿工很灵巧，跑过来给我们一人发了一根烟。

我低头看了一眼烟屁股上方的烫金字体。擦，耿工我实在不想戳穿你，求低调行吗，能不要把“1916年产的烟”拿出来吗?

我发现学姐又看了看耿工，继而看了看我，我头皮发麻。

进入正题。

文哥介绍了他现在的状况，我一听很有意思。原来文哥在东北有一个很大的粮油食品生产基地。目前他在上海成立了一个新公司进行相关货品的销售，在上海多家重量级超市提供粮油食品货源；公司建了十个左右的仓库，储存了从他基地运送过来的货物，由于一开始在拓展市场的时候把其中两个仓库连带货租给了一家起步不久 的同行业公司；出发点相对比较单纯，以一起发展为主要方针，然而该公司并没有沿袭文总的销售路线，而是在上海搞了一个电商销售平台，其中主营就是文总租给他两个仓库中的货源。

原先对电商完全不屑一顾的文哥竟然发现，租出去的这两个仓库在日营业额上竟然和他同等数量的实体销售达到了媲美的程度。并且对方只需2个员工加第三方快递就能搞定全部日常工作，而文哥至少需要4个人，同时还要2名司机起早贪黑的运送货物。

话说很多后起之秀都是从眼红开始的。文哥也不例外，对方用他的货竟然盘活了他以前一直不屑涉足的粮油电商市场。■

本文未完，更多内容请看原文：
http://developer.51cto.com/art/201304/391242.htm

PHP与MySQL通讯那点事

■ 在我们的一款WebGame的生产环境中，一次无意的strace抓包时，发现了php与mysql大量通讯的数据。这种情况，在游戏服务器刚启动时，是正常的，但如果是运行…

在我们的一款WebGame的生产环境中，一次无意的strace抓包时，发现了php与mysql大量通讯的数据。这种情况，在游戏服务器刚启动时，是正常的，但如果是运行一段时间之后，出现大量SELECT的SQL查询，绝对是有问题的，而且，所操作的数据库并不是配置库，那意味着，我们程序员的程序出现了违规的操作。具体结果大约如下：

如上图所示，php持续接收读取进程内描述符为3的响应包数据，描述符为3的为php与mysql建立的TCP通讯链接，这点也可以从313行的SELECT语句来确认。(原始数据丢失了，我模仿了一条。所以是配置库的SQL语句)

这是什么程序，想实现什么逻辑？为何要取这么多数据？

跟着这里的SELECT的sql语句，我定位到了相应的程序段：

```
/*  **  业务逻辑的代码  */
public function SItem($roleId,$baseId) {
    //...
    // ############写出下面这种代码的人都得死.##################
$this->dbrRole->select('*');
$this->dbrRole->from('role_items');
$this->dbrRole->where('role_id',$roleId);
$this->dbrRole->where('baseId',$baseId);
$result = $this->dbrRole->get()->row();
//看上去，这里好像正常，我们都以为框架会给我们只取一条。
//...
}
```

我们从代码上来看，好像明白程序员想根据对应的role_id到role_items表里取一条想符合的数据，所以，他调用了row方法，来取一条。看上去，这里好像正常，我们都以为框架会给我们只取一条。但实际上，框架是如何处理的呢？

我们来看下框架的对应row方法的实现过程。对了，我们是CodeIgniter框架的一个较老的版本。

```
/*
** 框架中，DB drive中，row相关方法的代码 **
*/
public function row($n = 0,$type = 'array'){
```

```
    if(!is_numeric($n)){
if(! is_array($this->_rowData)){
            $this->_rowData = $this-
    >rowArray(0);
}
if(isset($this->_rowData[$n])){
return $this->_rowData[$n];
        }
        $n = 0;
      }
     return ($type == 'object') ? $this-
    >rowObject($n) : $this->rowArray($n);
  }
  //继续跟进rowArray方法
 public function rowArray($n = 0){
      $result = $this->resultArray();
      if(count($result) == 0){
          return $result;
      }
       if($n != $this->_current &&
    isset($result[$n])){
          $this->_current = $n;
      }
      return $result[$this->_current];
 }
    //继续跟进resultArray方法 ###这个方法是重
    点###
public function resultArray(){
     if(count($this->resultArray) > 0){
          return $this->resultArray;
      }
       if(false === $this->resulter || 0 ==
    $this->recordCount()){
          return array();
      }
       $this->_dataSeek(0);
       while($row = $this->_fetchAssoc()){
          $this->resultArray[] = $row;
     //##########这个数组每次都增
    加_fetchAssoc()结果的内存大小数
}
      return $this->resultArray;
 }
//继续跟进_fetchAssoc方法
/*
** 对应driver的_fetchAssoc方法的代码
*/
protected function _fetchAssoc(){
      return mysql_fetch_assoc($this-
    >resulter);
  }
```

我们可以看到CodeIgniter框架的resultArray方法使用mysql(我们的php调用mysql的api用的是mysql函数，有点绕，后面解释)的mysql_fetch_assoc函数对缓冲区的数据进行遍历转换。将所有缓冲区的数据全部复制给$this->resultArray属性，再判断row方法中所需要的key的结果是否存在，再与返回的。

也就是说，框架层并没有只从mysql server(潜意识上的mysql server)那边取一条给我们调用者，而是取了所有结果，再返回一条。(先别

喷，后面解释) 当然，CI这种做法，也不是错。但我觉得有更好的改进方法。

这个问题，我们组的dietoad (征婚) 发现了这个问题，并给了修复方案。有些同学认为，这是程序员的错，程序员的SELECT语句没有加limit来限制条数。这我绝对赞同，而且，觉得写出这种代码的人都得死。

业务层：为这种业务需求的SQL语句加上limit限制

框架层：框架对于这种需求，自动控制，发现这种情况，直接返回1条

对于解决方案1，我写了一个正则，匹配select()方法被调用之后，row()方法被调用之前，中间没有使用limit()方法的所有代码，结果，发现量并不小。后来，我们决定两种方案同时实施，防止第二种出现漏掉的情况。

```
/*
** //改进为当_rowData不存在时，从_rowData
  的数量开始取，取小于$n条记录，避免 上面
  resultArray方法中从缓冲区取所有数据，复制双
  倍数据，占用内存的情况
*/
public function row ($n = 0, $type = 'array')
  {
    if(isset($this->_rowData[$n]))
    {
 return $this->_rowData[$n];
    }
    if (! is_numeric($n))
    {
      return $this->rowObject($n);
    }
     $ln=count($this->_rowData);
    //继续上次位置
while($ln++<=$n&&$r=$this->_
  fetchAssoc())
    {
      $this->_rowData[]=$r;
    }
    //需要几条就读几条
   //防止记录集为空报warning
    return isset($this->_
  rowData[$n])?$this->_
  rowData[$n]:array();
  }
```

在今年的4月末，鄙人写过另一篇关于CodeIgniter框架的设计缺陷问题，给我们游戏项目带来较大的影响，后来提交到github issues，并没得到回复，想了想，虽然官方的2.1.3版本中，也存在这个小问题。不过我觉得，这就不提交了，或许，我们的做法也符合他们的设计初衷。不过，我们还是在我们的项目中改进了。

如此改进之后，我们使用php的memory_get_usage()函数观察前后两个row()方法的结果时，果然发现内存使用情况有较大改善(改善幅度取决于SELECT的返回数据量)。

似乎，到这里就应该结束了，问题就这么被发现，被解决了。

但，我总觉得少了些什么呢？当我再次strace

抓包时，发现仍然存在大量的数据通讯，就像文章开头的那副截图一模一样。然而，这又是什么原因呢？

我顺手写了个内存占用的测试代码如下：

```
$db = mysql_connect('192.168.xx.xx','xxxx','xxxx');
 $sql = 'SELECT * from items';
 mysql_select_db('jv01',$db);
echo 'SELECT_DB: ',convert(memory_get_usage()),"\n";  //619.26 kb
 $r = mysql_query($sql,$db);  echo 'QUERY_SQL: ',convert(memory_get_usage()),"\n";    //619.98 kb ###什么？查询完之后，内存大小居然只增加了不到1k？我那个表可是几十M的数据啊
//sleep(50);  // hold住进程，别销毁，留着看当前进程的内存分配1
$arr = array();
while ($rs = mysql_fetch_assoc($r))
{
    $arr[]=$rs;
}
echo 'FETCH_RS: ',convert(memory_get_usage()),"\n";    //27.11 mb  ###什么？刚刚不是只增加了1k吗？这里的遍历的结果集怎么突增几十M啊？尼玛这到底是什么情况？
 unset($arr);  echo 'UNSET: ',convert(memory_get_usage()),"\n";
 //620.12 kb  #### $arr z占了几十M
 mysql_free_result($r);  echo 'FREE_R: ',convert(memory_get_usage()),"\n";
 //620 kb    ### 结果集居然只有0.12 k？这不扯淡么？ 莫非。。。莫非缓冲区的数据php统计不到？莫非不是调用zend 内存申请函数来申请内存的？
//sleep(50);  // hold住进程，别销毁，留着看当前进程的内存分配2
function convert($size)
{
 $unit=array('b','kb','mb','gb','tb','pb');
return @round($size/pow(1024,($i=floor(log($size,1024)))),2).' '.$unit[$i];
}
/*  //返回结果如下：
SELECT_DB: 619.26 kb
QUERY_SQL: 619.98 kb
FETCH_RS: 27.11 mb
UNSET: 620.12 kb
FREE_R: 620 kb  */
```

看到结果时，我不禁XX一紧，什么？这你妈什么情况？查询完之后，内存大小居然只增加了不到1k？我那个表可是几十M的数据啊？遍历结果集之后，怎么突增几十M啊？尼玛这到底是什么情况？strace返回的大量数据到底存在哪的？算不算php进程申请的？

后来，我再次执行如上程序，再定时用free、/proc/PID/maps 之类系统工具，查看系统的内存使用情况，确认了当前进程的内存占用确实存在。那么可能的情况就是memory_get_usage()函数并没有获取到 mysql_query之后的内存占用情况。由于比较怀疑，未学跟进了memory_get_usage()函数的源码■

本文未完，更多内容请看原文：
http://developer.51cto.com/art/201304/392128.htm

创业像水墨画：三千世界 致在桥上看风景的你

■ 一部《致青春》的影片让初次担任导演的赵薇大大赚足了票房，一时间，无论是即将走出象牙塔的新一届莘莘学子，还是三十而立四十不惑，抑或是五十知天命的职场木头人们都开始回忆并反思自己的青春之路。

一部《致青春》的影片让初次担任导演的赵薇大大赚足了票房，一时间，无论是即将走出象牙塔的新一届莘莘学子，还是三十而立四十不惑，抑或是五十知天命的职场木头人们都开始回忆并反思自己的青春之路，更有甚者，曾放出“那些长的好看的人才有青春，像我这样的就只有大学了”的自嘲言论。遗憾也好，欣慰也罢，青春似乎嗖的一下就被我们远远甩在了身后，来不及细想，没有时间回味。

也许有人说，青春就是用来浪费的。也有人说，就要趁着年轻多折腾。这样，也便有了不甘于平庸平淡、不愿受限于朝九晚五的作息、不满足于两平米的办公格子间的“不安分”的心。创业，也便成为了他们想要证明自身价值和实现青春梦想的最响铮铮的口号。那么，是否每个人都适合创业？创业者要具备哪些基本的素质？创业过程中一般会遇到哪些阻碍和瓶颈？一些成功的创业者是否有一些共通的东西可以与我们分享？

想创业，你是否具备这些。。。

俗话说，巧妇难为无米之炊。因此，一个好的创业方向、点子或创意至关重要，不然，空有一腔创业的热情和一个不甚清晰的行为目标以及“走着看不好就换”的逃兵心态是很难实现创业的初衷的。在笔者的观念里，做事情就要“不打无准备的仗”，虽然创业多是摸着石头过河，同时还可能加以机遇的光顾青睐，但前期想清楚自己要做什么、大概怎么做、预期效果如何等的这样一个基本的经营思路是很必须的。

一般对创业成功者的总结有两个基本要素，那就是热情和坚持。其实这是做任何事情想要成功的基本通用特点。最近追了一部去年底出品的一部连续剧《温州一家人》，原名《创业年代》。我没有接触过本源的温州人，并不深深触及电视剧里面所谓的温州人身上所流淌着的“做生意、创业、赚大钱”的血液。虽然我自认为也不是太安分守己的人，时不时的迸出点新想法或者有立志自己做点事儿的念头，但我还是不得不佩服剧里面的周万顺“太能折腾了“，为了赚大钱，在80年代冒着大逆不道的恶名卖掉老祖屋，走出古树村，孤注一掷发大财。捡过破烂、卖过皮鞋、倒腾过开关、开过皮鞋厂、采掘过石油，露宿过街头、睡过拖拉机、扎根过住牲畜的破窑洞，几度身无分文、数次债台高筑、几次挣扎寻求解脱。。。所幸的是，他的热情和坚持终于让他成为了中国的石油大亨洛克菲勒。

热情很重要，坚持必不可少。但是仅这两样，也是绝对不够的。我们不可否认周万顺的会做人和厚道给他不断带来的创业机会，同时我们更不能否认他对这些机会的敏锐捕捉和把握；我们不能否认四眼老师和棠梨头在他开皮鞋厂时这些合作伙伴，我们也更不能磨灭家人尤其是媳妇儿在他背后的支持和安慰；我们不能抹杀那个时代背景下可以赊销的商业信任，我们也更不能忘记在紧要关头卖厂子卖房子和在国外女儿的强大资金支持。。。在这个坎坷曲折的创业经历中，具备良好的心理素质也是影响成功与否的一个关键性隐含因素，因为每一次的煎熬和欣喜，你从来都不知道是否会有明天。

创业过程中变数很大，一个环节出错，都有可能出现《致青春》里面陈孝正所谓的“我的人生是只能建造一次的大厦，必须精确无比，一厘米的误差都可能使它坍塌。“成功不可复制，却有经验可以共同分享。我们邀请了TEEKER.com的创始人蓝灿辉、七牛云存储创始人许式伟，以及PHPCMS、CmsTop创始人钟胜辉三位在互联网创业方面的实践者，让他们和我们51CTO的读者共同探讨和分享创业的话题。

做自己喜欢的事儿更容易成功

都说“兴趣是最好的老师“，创业也是一样，如果想清楚自己一定要创业，何不遵从自己的内心趁着年轻去搏一把？正因为年轻，即使失败了，还有时间和机会让自己重来，年轻时的失败都不叫作失败，这是一笔成长的财富。创业，没有年龄界限的区隔和分别，也曾访过50岁开始创业的，也曾见过七老八十起步的，重要的是，你是否具备一颗喜欢和坚韧的心。

“目前我正在做自己喜欢做的事，“TEEKER.com的创始人蓝灿辉说，”有明确的目标，战略也很清晰，虽然有不少的困难，但我坚信坚持下去就会成功，这个信念也是我最大的动力。“同时，他认为，创业已经成为了他的唯一职业选择，因为他觉得只有通过创业才能实现他的人生目标，”坚持创业，坚持下去“是他对自己的要求。

“这两年来，我比以往任何一个时候都感觉到激情和充实，我喜欢这种感觉。” 七牛云存储创始人许式伟说，“想创业，最主要得原因是希望能够做成一件伟大的事情，但做成一件伟大的事情并不一定非要选择创业，也可以在大公司做，比如我原先认为在盛大创新院可以让我达成所愿，但我先后在金山、百度和盛大都工作过，个人经验表明，大公司有大公司的瓶颈，除非你的乐趣与这个大公司的主营业务匹配，否则所能发挥的空间是非常受限的，所以后来我从盛大离职，腾讯和360向我抛来橄榄枝的时候，我还是坚定的选择了创业。“

“我喜欢按照自己的想法做事儿的感觉，苦中有乐。“PHPCMS、CmsTop创始人钟胜辉回忆说，”从小家里穷，交不起学费，学校不发书，中途还被撵回家要学费，这让我在同学面前很难堪，所以从小我就总想证明自己比别人强。上大学后开始接触互联网，很喜欢，喜欢研究web技术，这样就可以按照自己的想法去做互联网。自从对互联网着迷后，我就对专业课一点兴趣都没有了，天天泡机房和网吧，后来干脆停学创业了。“钟胜辉自我剖析道，我的创业激情源于兴趣和不断证明自己。对于自己认同和喜欢的事情，很执着；对自己不喜欢的事情，我是一点精力都不愿意浪费。我最近也在思考自己真正想要的是什么，我认为人到一定的阶段后，要有所为有所不为，要追寻梦想，找到自己喜欢的工作和生活状态，快乐工作，快乐生活，最大化发挥自我价值，最大化创造社会价值。

曾遭遇了哪些苦难

一帆风顺只是美好的寄语，民间有句老话“头三脚难踢“，意即万事开头难，创业初期“三五个人十来杆枪”作坊式没日没夜的劳作，甚至时刻提着胎死腹中或过早夭折的为前景预期担忧，心一直在嗓子眼上晃荡的滋味，个儿唯有自知。

“我的两个最大的苦难是资金和人才。” 蓝灿辉说，资金的问题我通过在亲朋好友中融资、借贷，再加上自己原有的积累，虽然经常很紧张，但基本都应付过来了。在人才问题上，通过对项目的合理规划，清晰的战略吸引了早期参与者。在团队中实行广泛的期权和股权激励，也成功实现了人才的引进和保持。

许式伟认为七牛的发展总体来说，还是比较顺利的，很多事情都是水到渠成，但是也曾面临找人、产品方向和建立信任的困难。“在找人方面，七牛选择了云存储这样一个看似是巨头才能玩的游戏。刚创业的时候，朋友圈里面看好的不多，挺长的一段时间人员招聘上比较困难。我们的想法是选择在熟人圈里找人。有时对方犹豫，我就会说，也许你觉得方向不好，但请相信方向可以探索，最重要的是整个团队非常靠谱。整个团队都是都是熟人，配合起来非常顺畅。”许式伟回忆说，七牛最早成立的时候是5个人，主要来自盛大和金山，或者盛大金山双重背景。3个月，我们从零开始，完成了Q盘（QBox）的服务器、Web端、Android端、iPhone端、iPad端、Windows端（因为是基于wxWindows+Golang，故方案上是跨平台的，当时内部有Mac下的演示版），如果不是方向调整，再有半个月我们就可以发布Mac和Linux版本，这种开发效率，直到现在我仍然觉得不可思议，并以此为自豪。

“第二个问题是产品方向问题。我们最初选择了做面向个人的存储，即网盘，就在我们把Q盘产品做出来，准备推广的时候，我们敏锐的意识到整个市场的前进方向在往对七牛非常不利的方向发展。整个七牛初创团队每个人个性极强，善于反思，在了解到Camera 360评估第三方云存储方案时，我们专门跑了一趟成都确立了合作事宜，由七牛为Camera 360提供云存储服务。而七牛悄无声息的转战企业云存储市场，紧接着七牛拿下了Weibo+。这是我们最初的两大客户。”

“第三个问题是如何建立信任。事实上企业云存储方向并不容易，尤其是对初创团队来说，凭什么让客户信任你，这相当困难。因此有一天我们也要宴请七牛最初十大客户，这叫感恩。我们用一种把事情做到极致，让每一个选择我们的客户都满意的决心，七牛逐步在企业客户间赢得了口碑。”

“我坚信困难是可以克服的，未来是可以创造的。”钟胜辉信心满满，“没技术，我就天天泡学校图书馆、机房和网吧，熬夜学技术。没钱，我就学技术，给企业做网站赚钱，后来做虚拟主机业务采用分期付款，业务做起来有收入了就滚动式发展。没管理经验，我就看文章和书籍，找人指教，当然更多的是从实践中思考和总结，悟出来了，自然就能积累经验，管理没有捷径，就是多实践、多总结、多领悟。”

经验与建议分享

听人劝，吃饱饭。成功没有捷径，同样，创业也没有捷径，只有不断摸索、经历和感悟，但是却可以有成功者的心得与感悟拿来分享和借鉴，■

本文未完，更多内容请看原文：
http://developer.51cto.com/art/201305/393118.htm

编者按

泛型是程序设计语言的一种特性。允许程序员在强类型程序设计语言中编写代码时定义一些可变部分，那些部分在使用前必须作出指明。下面，各位网友们认真看看30分钟，完全掌握泛型的用法。

30分钟泛型教程

我们先来看一个最为常见的泛型类型List<T>的定义

(真正的定义比这个要复杂的多，我这里删掉了很多东西)

```
[Serializable]
public class List<T> : IList<T>,
  ICollection<T>, IEnumerable<T>
 {
    public T this[int index] { get; set; }
    public void Add(T item);
    public void Clear();
    public bool Contains(T item);
    public int IndexOf(T item);
    public bool Remove(T item);
    public void Sort();
    public T[] ToArray();
 }
```

List后面紧跟着一个<T>表示它操作的是一个未指定的数据类型（T代表着一个未指定的数据类型）

可以把T看作一个变量名，T代表着一个类型，在List<T>的源代码中任何地方都能使用T。

T被用作方法的参数和返回值。

Add方法接收T类型的参数，ToArray方法返回一个T类型的数组。

注意：

泛型参数必须以T开头，要么就叫T，要么就叫TKey或者TValue；

这跟接口要以I开头是一样的，这是约定。

下面来看一段使用泛型类型的代码：

```
var a = new List<int>();
 a.Add(1);
 a.Add(2);
 //这是错误的，因为你已经指定了泛型类型为int，就不能在这个容器中放入其他的值
//这是编译器错误，更提升了排错效率，如果是运行期错误，不知道要多么烦人
a.Add(“3”);
var item = a[2];
```

请注意上面代码里的注释

二、泛型的作用(1)：

作为程序员，写代码时刻不忘代码重用。

代码重用可以分成很多类，其中算法重用就是非

常重要的一类，假设你要为一组整型数据写一个排序算法，又要为一组浮点型数据写一个排序算法，如果没有泛型类型，你会怎么做呢?

你可能想到了方法的重载。

写两个同名方法，一个方法接收整型数组，另一个方法接收浮点型的数组。

但有了泛型，你就完全不必这么做，只要设计一个方法就够用了，你甚至可以用这个方法为一组字符串数据排序。

三、泛型的作用(2)：

假设你是一个方法的设计者，这个方法需要有一个输入参数，但你并能确定这个输入参数的类型，那么你会怎么做呢?

有一部分人可能会马上反驳：“不可能有这种时候！”

那么我会跟你说，编程是一门经验型的工作，你的经验还不够，还没有碰到过类似的地方。

另一部分人可能考虑把这个参数的类型设置成Object的，这确实是一种可行的方案，但会造成下面两个问题，如果我给这个方法传递整形的数据（值类型的数据都一样），就会产生额外的装箱、拆箱操作，造成性能损耗。

如果你这个方法里的处理逻辑不适用于字符串的参数，而使用者又传了一个字符串进来，编译器是不会报错的，只有在运行期才会报错。

(如果质管部门没有测出这个运行期BUG，那么不知道要造成多大的损失呢)

这就是我们常说的：类型不安全。

四、泛型的示例：

像List<T>和Dictionary<TKey,TValue>之类的泛型类型我们经常用到，下面我介绍几个不常用到的泛型类型。

ObservableCollection<T>

当这个集合发生改变后会有相应的事件得到通知。

请看如下代码：

```
static void Main(string[] args) {
  var a = new ObservableCollection<int>();
  a.CollectionChanged += a_
  CollectionChanged;
 }
  static void a_CollectionChanged(object sen
  der,NotifyCollectionChangedEventArgs e)
{
  //可以通过Action来判断是什么操作触发了事件
  //e.Action ==
  NotifyCollectionChangedAction.Add
 //可以根据以下两个属性来得到更改前和更改后的
  内容
 //e.NewItems;
 //e.OldItems;
 }
```

使用这个集合需要引用如下两个名称空间

```
using System.Collections.ObjectModel;
using System.Collections.Specialized;
```

BlockingCollection<int>是线程安全的集合

来看看下面这段代码

```
var bcollec = new
   BlockingCollection<int>(2);
//试图添加1-50
 Task.Run(() =>  {
//并行循环
   Parallel.For(1, 51, i =>
{
 bcollec.Add(i);
       Console.WriteLine("加入：" + i);
    });
  });
Thread.Sleep(1000);
Console.WriteLine("调用一次Take");
bcollec.Take();
//等待无限长时间
Thread.Sleep(Timeout.Infinite);
```

输出结果为：

加入：1

加入：37

调用一次Take

加入：13

BlockingCollection<int>还可以设置CompleteAdding和IsCompleted属性来拒绝加入新元素。

.NET类库还提供了很多的泛型类型，在这里就不一一例举了。

五、泛型的继承：

在.net中一切都继承字Object，泛型也不例外，泛型类型可以继承自其他类型。

来看一下如下代码

```
public class MyType  {
   public virtual string getOneStr()
    {
       return "base object Str";
    }
 }
public class MyOtherType<T> : MyType
 {
    public override string getOneStr()
    {
       return typeof(T).ToString();
    }
}
class Program
  {
   static void Main(string[] args)
   {
MyType target = new MyOtherType<int>();
       Console.WriteLine(target.getOneStr());
  Console.ReadKey();
    }
}
```

泛型类型MyOtherType<T>成功的重写了非泛型类型MyType的方法。如果我试图按如下方式从MyOtherType<T>类型派生子类型就会导致编译器错误。■

本文未完，更多内容请看原文：
http://developer.51cto.com/art/201305/392463.htm

成人网站性能提升20倍之经验谈

□ 编译：@老码农的自留地

色情业是个大行业。互联网上没有多少网站的流量能和最大的色情网站相匹敌。

要搞定这巨大的流量很难。更困难的是，在色情网站上提供的很多内容都是低延迟的实时流媒体而不是简单的静态视频。但是对于所有碰到过的挑战，我很少看到有搞定过它们的开发人员写的东西。所以我决定把自己在这方面的经验写出来。

问题是什么?

几年前，我正在为当时全世界访问量排名26的网站工作 — 这里不是说的色情网站排名，而是全世界排名。

当时，该网站通过RTMP(Real Time Messaging protocol)协议响应对色情流媒体的请求。更具体地说，它使用了Adobe的FMS(Flash Media Server)技术为用户提供实时流媒体。基本过程是这样的：

1.用户请求访问某个实时流媒体

2.服务器通过一个RTMP session响应，播放请求的视频片段

3.因为某些原因，FMS对我们并不是一个好的选择，首先是它的成本，包括了购买以下两者：

4.为每一台运行FMS的服务器购买Windows的版权

大约4000美元一个的FMS特定版权，由于我们的规模，我们必须购买的版权量数以百计，而且每天都在增加。

所有这些费用开始不断累积。撇开成本不提，FMS也是一个比较挫的产品，特别是在它的功能方面（我过一会再详细说这个问题）。所以我决定抛弃FMS，自己从头开始写一个自己的RTMP解析器。

最后，我终于把我们的服务效率提升了大约20倍。

开始

这里涉及到两个核心问题：首先，RTMP和其他的Adobe协议及格式都不是开放的，这就很难使用它们。要是对文件格式都一无所知，你如何能对它进 行反向工程或者解析它呢？幸运的是，有一些反向工程的尝试已经在公开领域出现了（并不是Adobe出品的，而是osflash.org，它破解了一些协议），我们的工作就是基于这些成果。

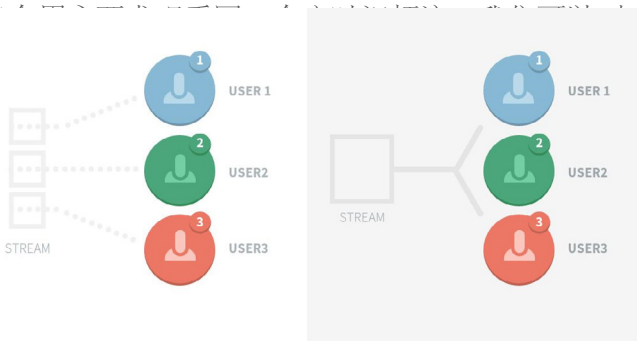
注：Adobe后来发布了所谓的“规格说明书”，比起在非Adobe提供的反向工程wiki和文档中披露的内容，这个说明书里也没有啥新东西。他们 给的规格说明书的质量之低劣达到了荒谬的境地，近乎不可能通过该说明书来使用它们的库。而且，协议本身看起来常常也是有意做成具有误导性的。例如：

1.他们使用29字节的整形数。

2.他们在协议头上所有地方都采用低地址存放最高有效字节(big endian)的格式，除了在某一个字段（而且未标明）上采用低地址存放最低有效字节(little endian)的格式。

3.他们在传输9K的视频时，不惜耗费计算能力去压缩数据减少空间，这基本上是没意义的，因为他们这么折腾一次也就是减少几位或几个字节，对这样的一个文件大小可以忽略不计了。

还有，RTMP是高度以session为导向的，这使得它基本上不可能对流进行组播。理想状态下，如果



我的解决办法

想到了这些，我决定把典型的响应流重新打包和解析为FLV“标签”（这里的“标签”指某个视频、音频或者元数据）。这些FLV标签可以在RTMP下顺利地传输。

这样一个方法的好处是：

我们只需要给流重新打包一次（重新打包是一个噩梦，因为缺少规格说明，还有前面说到的恶心协议）。

通过套用一个FLV头，我们可以在客户端之间顺畅地重用任何流，而用内部的FLV标签指针（配以某种声明其在流内部确切位置的位移值）就可以访问到真正的内容。

我一开始用我当时最熟悉的C语言进行开发。一段时间后，这个选择变得麻烦了，所以我开始学习Python并移植我的C代码。开发过程加快了，但在做 了一些演示版本后，我很快遇到了资源枯竭的问题。Python的socket处理并不适合处理这些类型的情况，具体说，我们发现在自己的Python代码里，每个action都进行了多次系统调用和context切换，这增加了巨大的系统开销。

改进性能：混合使用Python和C

在对代码进行梳理之后，我选择将性能最关键的函数移植到内部完全用C语言编写的一个Python模块中。这基本是底层的东西，具体地说，它利用了内核的epoll机制提供了一个O(log n)的算法复杂度。

在异步socket编程方面，有一些机制可以提供有关特定socket是否可读/可写/出错之类的信息。过去，开发人员们可以用select()系 统调用获取这些信息，但很难大规模使用。Poll()是更好的选择，但它仍然不够好，因为你每次调用的时候都要传递一大堆socket描述符。

Epoll的神奇之处在于你只需要登记一个socket，系统会记住这个特定的socket并处理所有内部的杂乱的细节。这样在每次调用的时候就没 有传递参数的开销了。而且它适用的规模也大有可观，它只返回你关心的那些socket，相比用其他技术时必须从10万个socket描述符列表里挨个检查 是否有带字节掩码的事件，其优越性真是非同小可啊。■

本文未完，更多内容请看原文：
http://developer.51cto.com/art/201305/393079.htm